MASTER EN SCIENCES INFORMATIQUES

Du DES au RIJNDAEL: une étude de standards de chiffrement

Fravezzi, Yves

Award date:
2001

Facultés Universitaires Notre-Dame de la Paix, Namur

Institut d'Informatique.

Année académique 2000-2001

Du DES au RIJNDAEL

une étude de standards de chiffrement.

Yves Fravezzi

Mémoire présenté en vue de l'obtention du grade de Licencié en Informatique

RÉSUMÉ - SUMMARY

L'algorithme cryptographique public et gratuit DES a régné sur la planète depuis 1977 malgré les révisions quinquennales. En 2000, lors de la dernière révision, l'algorithme conçu conjointement par deux candidats belges, Messieurs Vincent Rijmen et Joan Daemen a été sélectionné parmi les 15 candidats de 12 pays différents. Ce nouvel algorithme baptisé RIJNDAEL, remplacera dorénavant le DES. Le présent mémoire étudie les mérites respectifs et l'efficacité de ces deux systèmes. L'auteur s'est attaché essentiellement à aborder les critères de choix du NIST (National Institute of Standards and Technology) à savoir la solidité cryptographique, l'efficacité et d'autres critères tels que la clarté des fondements mathématiques, la clarté des spécifications, etc. ainsi que les précautions à prendre lorsqu'il s'agit d'implémenter cet algorithme.

L'analyse du RIJNDAEL fait apparaître de façon manifeste les qualités de cet algorithme qui lui assureront vraisemblablement un règne de plusieurs décennies.

The public and royalty-free algorithm DES has taken the first place in the world since 1977 in spite of the quinquennial revisions. In 2000, while the last revision took place, the algorithm created by two Belgians Mr Vincent Rijmen and Mr Joan Daemen has been chosen among 15 appliants from 12 different countries. This new Belgian algorithm, named RIJNDAEL, is going to take over the place of DES. This study analyses the value and the efficiency of both systems.

The author put his main effort in the approach to the evaluation criteria of the NIST (National Institute of Standards and Technology) i.e. the cryptographic soundness, the efficiency and other criteria like the clarity of the mathematical basics, the accuracy of the specifications...as well as the attention to pay when the algorithm has to be implemented.

The analysis of the RIJNDAEL points out the qualities of this algorithm likely insuring a rule during several decades.

Remerciements

Nombreux sont ceux et celles qui, inlassablement m'ont encouragé et aidé à la réalisation de ce mémoire.

Je voudrais exprimer ma reconnaissance à Monsieur le professeur J. Ramaekers, pour ses conseils avisés lors de l'élaboration de ce travail.

Ma gratitude va également à Monsieur V. Rijmen, dont les conseils et les compétences m'ont permis de progresser utilement dans mes recherches.

J'adresse ma plus vive reconnaissance à mes amis, pour leurs encouragements et leur disponibilité.

Merci à Dominique Champluvier, botaniste et au Colonel Van Hammée pour leur soutien linguistique, ainsi qu'au lieutenant Olivier Thonnard pour ses conseils judicieux.
Enfin, merci au Commandant J-J Stoffel pour son soutien de tous les instants.

1. INTRODUCTION

Depuis toujours, l'homme a fait usage de la cryptographie pour la protection de ses communications militaires et diplomatiques.
L'usage de la cryptographie s'est ensuite répandu d'une manière foudroyante avec l'emploi généralisé de l'outil informatique dans tous les domaines de la vie du citoyen.

Les besoins sont maintenant multiples et différents ; ils s'étendent dans des domaines variés tels la protection des données à caractère privé, le secret médical, la sécurité des paiements électroniques, la protection des données sur des réseaux locaux... Une des conséquences immédiates en est une augmentation de la masse des données à chiffrer.

De la même manière, les tentatives de « craquer » le chiffrement, d'attaquer les messages chiffrés pour en découvrir l'information en clair se sont développées ; les méthodes sont multiples et nombreuses. Le marché est juteux et souvent impuni voire non punissable car les instruments légaux ne sont pas adaptés. Les moyens et les outils pour réaliser ces attaques sont de plus en plus puissants et peuvent être partagés.

L'objectif du présent mémoire est de décrire, de manière simple, les critères de choix d'un « bon » algorithme, tels ceux qui ont poussé le NIST (National Institute of Standards and Technology) à choisir comme nouveau standard l'algorithme RIJNDAEL et plus spécifiquement à développer le critère de la solidité cryptographique en l'appliquant à ce nouveau standard universel.

L'objectif est également de pouvoir passer en revue les principaux aspects qu'un informaticien doit prendre en considération lorsqu'il s'agit de choisir et « d'implémenter » un algorithme cryptographique. Le mémoire reprendra un rappel historique dans lequel les notions principales de la cryptographie seront évoquées.
Il s'agit de concepts qui sont encore d'application à l'heure actuelle : la notion de confusion et de diffusion de l'information et les thèmes tels que l'entropie et la distance d'unicité concepts développés dans la théorie de Shannon qui a influencé d'une manière fondamentale la cryptographie contemporaine et donc les concepteurs des algorithmes modernes.

Le chapitre 3 sera axé principalement sur le DES (Data Encryption Standard) et ses évolutions. Depuis plusieurs décennies et encore à l'heure actuelle, le DES fut en effet l'algorithme le plus répandu au monde.
Il décrira les principes fondamentaux de ce système Crypto et une des attaques les plus révolutionnaires de la cryptographie moderne : l'attaque différentielle.

Deux années durant, les sommités cryptographiques mondiales se sont appliquées à analyser les systèmes cryptographiques candidats à la relève du DES élu tous les 5 ans depuis 1977 standard AES[1] par le NIST (The National Institute of Standards and Technology).
Quelques 15 algorithmes provenant de 12 pays différents étaient candidats.
Un algorithme belge conçu par Vincent Rijmen et Joan Daemen a été choisi. Le RIJNDAEL a été retenu le 2 octobre 2000 comme nouveau standard AES.

Le RIJNDAEL est amené à supplanter le DES et à être utilisé mondialement.
C'est pourquoi le chapitre suivant et, en fait, l'effort principal du présent mémoire lui seront consacrés.
Nous y étudierons en détail toutes les étapes de cet algorithme. Nous y tenterons ensuite de déterminer les critères de sécurité qui ont influencé les concepteurs de ce système.
Nous parlerons des différentes attaques que l'opposant peut tenter d'appliquer sur cet algorithme pour découvrir l'information et la valeur de l'effort à consacrer pour les réaliser. Nous comparerons ces valeurs avec celles du DES et éventuellement avec d'autres algorithmes candidats.

Parce que l'efficacité est également un critère dont il faut tenir compte, le chapitre 6 sera consacré à la performance des algorithmes. Nous y verrons en bref comment l'algorithme peut être utilisé sur des plates-formes différentes et quelles sont les valeurs à prendre en considération pour évaluer la performance d'un algorithme.

Le dernier chapitre traitera d'autres notions à prendre en considération lorsqu'il s'agit « d'implémenter » un algorithme. Nous parlerons de timing attack et de power attack.

A l'issue de ce mémoire, nous visons à avoir dégagé les critères les plus importants de choix d'un algorithme et avoir montré dans quelle mesure le nouveau standard est le choix judicieux. Nous donnerons également des idées de grandeur dans le cadre de ce type d'évaluation.

Au cours de la lecture, des limitations à la profondeur de l'étude apparaîtront. Ce mémoire n'est pas dédié aux cryptographes ni aux mathématiciens passionnés de cryptographie qui n'y trouveront pas les développements mathématiques que les créateurs de système de chiffrement ont mis en œuvre pour concevoir leurs algorithmes. Ceci est dû à la volonté de circonscrire l'ouvrage à des données essentiellement pratiques.

[1] Advanced Encryption Standard

2. UN PEU D'HISTOIRE...

La cryptographie n'est certes pas nouvelle, mais elle connaît un développement extraordinaire depuis la deuxième moitié du 20ème siècle.
Après une brève description des mécanismes qui ont régi la cryptographie classique, nous passerons en revue les principaux concepts de la cryptographie moderne qui ont modifié l'histoire de cette science.
Il s'agit essentiellement de la théorie de Shannon et des principes de confusion et de diffusion qui sont encore aujourd'hui les soucis essentiels des concepteurs d'algorithmes cryptographiques.

2.1. Cryptographie Classique

Avant l'avènement des ordinateurs, la cryptographie classique utilisait principalement deux mécanismes afin de rendre les textes inintelligibles. Il s'agit de la transposition et de la substitution. Nous tenterons dans un premier temps de décrire ces deux mécanismes et dans la suite du mémoire, nous verrons dans quelles mesures ils sont encore d'actualité.
Pour chacun de ces mécanismes, nous tenterons de les définir d'une manière informelle et ensuite plus formelle et, par des exemples simples, d'en induire les principes et critères de qualité.

2.1.1. La substitution

2.1.1.1. Définition

La substitution consiste à remplacer chaque caractère du texte en clair par un autre caractère, la correspondance est obtenue en respectant une règle appropriée. Il peut s'agir d'un glissement prédéterminé (la clé) sur une table de référence (l'alphabet en l'occurrence) ou d'un mécanisme plus complexe.

Plus formellement,

Si P est l'ensemble des textes en clairs possibles,
C l'ensemble des textes chiffrés,
K l'ensemble des clés possibles,

Pour chaque clé k de K,
$\exists$ E_k, une règle de chiffrement (par substitution) de $P \rightarrow C$ et
D_k, une règle de déchiffrement de $C \rightarrow P$
telles que $D_k (E_k (x)) = x$ (x appartenant à P)

2.1.1.2. Exemple 1 : la substitution par décalage – chiffrement de César

la substitution simple par décalage s'exprime dans Z_{26} de la manière suivante :

$E_k(x) = x + k \bmod 26$

et le déchiffrement de la même manière

$D_k(y) = y - k \bmod 26$

Prenons par exemple

```
A B C D E F G H I J K  L  M  N  O  P  Q  R  S  T  U  V  W  X  Y  Z
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

Substituons en prenant la clé de chiffrement appelée 'le chiffrement de Jules César', c'est-à-dire k = 3 dans l'alphabet modulo 26 (a donne d, b correspond à e etc..)

Et le texte en clair suivant

```
lessanglotslongsdesviolons
```

Ce texte est converti en chiffres :

```
11 4 18 18 0 13 6 11 14 19 18 11 14 13 6 18 3 4 18 21 8
14 11 14 13 18
```

la substitution est opérée en ajoutant 3 (mod 26) :

```
14 7 21 21 3 16 9 14 17 22 21 14 17 14 9 21 6 7 21 24 11
17 14 17 16 21
```

Et devient finalement en texte

```
ohvvdqjorwvoioivghvylrorqv
```

2.1.1.3. Critères de qualité

Deux critères de qualité apparaissent immédiatement :

1) l'opération de chiffrement doit être faisable facilement, c'est-à-dire partout définie et ne nécessitant pas trop d'opérations ;

2) l'opposant (ou le cryptanalyste) doit avoir un maximum de difficulté à déchiffrer en essayant toutes les clés possibles (recherche exhaustive).

Cette deuxième condition n'est pas suffisamment remplie dans le cas du chiffrement décrit plus haut ; l'espace K est en effet trop petit.

En faisant l'essai sur toutes les clés possibles, l'opposant obtiendra rapidement la solution.

Texte	clé
ohvvdqjorwvoioivghvylrorqv nguucpinqvunhnhufguxkqnqpu mfttbohmputmgmgteftwjpmpot lessanglotslongsdesviolons	0 1 2 3

Finalement la recherche peut s'arrêter lorsque k= 3
Le nombre moyen de déchiffrements successifs est de 13.

2.1.1.4. Exemple 2 : Une substitution polyalphabétique

La substitution peut prendre des formes plus complexes. Voici un exemple d'une substitution polyalphabétique utilisant un mot code.Cet exemple est une méthode de chiffrement bien connue appelée le chiffrement de Blaise de Vigenère[2].

En utilisant la même méthode que pour le système de chiffrement décrit plus haut, nous allons maintenant utiliser un mot code d'une longueur n et ajouter au texte en clair le mot code de manière répétitive. Notre exemple prendra alors la forme suivante :

Choisissons le mot code 'merlin' (12 4 17 11 8 13)

le texte chiffré deviendra alors :

```
11 4 18 18 0 13  6 11 14 19 18 11 14 13  6 18  3  4 18 21  8 14 11 14 13 18
12 4 17 11 8 13 12  4 17 11  8 13 12  4 17 11  8 13 12  4 17 11  8 13 12  4
-----------------------------------------------------------------------------
23 8  9  3 8  0 18 15  5  4  0 24  0 17 23  3 11 17  4 25 25 25 19  1 25 22
```

soit

lessanglotslongsdesviolons

est chiffré en

xijdiaspfeayarxdlrezzztbzw

Le nombre de clés possibles s'élève maintenant à 26^n. Toutefois, si l'on connaît n, une attaque sur toutes les clés possibles reste tout à fait réalisable en peu de temps sur un ordinateur d'aujourd'hui.

[2] Blaise de Vigenere vivait au 16ème sciècle.

2.1.1.5. Exemple 3 : Une substitution aléatoire

Il peut s'agir également de remplacer les caractères du texte clair en établissant une table de correspondance entre les caractères aléatoires. La clé est évidemment plus longue, 26 caractères dans ce cas.

Prenons par exemple la table de correspondance suivante :

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
T Q E K A N I Y H G C D M P J L V F B W R O Z S U X
```

Le texte que nous avons utilisé précédemment deviendra

```
lessanglotslongsdesviolons
```

```
dabbtpidjwfdjpibkabohjdjpb
```

Cette fois, une attaque sur toutes les clés possibles semble dans ce cas plus difficile voire impossible dans des temps raisonnables ; le texte en clair apparaîtra après un nombre relativement élevé d'itérations (en moyenne ½.26!).

2.1.2. La transposition ou permutation

2.1.2.1. Définition

Dans un système à transposition, l'idée est de garder les caractères du texte clair et de **modifier leur position**.

Plus formellement la permutation peut se définir de la manière suivante :

Soit un entier $m \in N$, P et C dans $(Z_{26})^m$
Et K, l'ensemble des permutations de {1,..,m}

$E_k(x_1,\ldots,x_m) = (x_{k(1)}, \ldots, x_{k(m)})$
et
$D_k(y_1,\ldots,y_m) = (y_k{}^{-1}{}_{(1)}, \ldots, x_k{}^{-1}{}_{(m)})$

où k^{-1} est la permutation inverse de k

2.1.2.2. Exemple de permutation

Prenons un texte en clair :

```
l'attaque aura lieu à midi par le versant sud ouest
```

et la permutation suivante avec m=7

1	2	3	4	5	6	7
6	3	1	7	4	2	5

le texte se découpe en groupes de 7 lettres :

`lattaqu | eaurali | euamidi | parleve | rsantsu | douestt`

se transformera en

`tqaault | ulaaier | aduiiem | rvaeepl | assturn | utostde`

et le texte chiffré devient

`tqaaultulaaieraduiiemrvaeeplassturnutostde`

Plus m est grand, plus le cryptanalyste aura de difficultés à découvrir le texte en clair. Une recherche exhaustive ou brutale (sur l'ensemble des clés possibles) nécessite en moyenne m!/2 essais.

2.2. La cryptographie contemporaine

Au cours de cette section, nous évoquerons quelques principes fondamentaux de la cryptographie contemporaine.

Parce que la cryptographie n'a de sens que si l'on veut se protéger contre des obtentions non autorisées d'informations, c'est-à-dire s'il existe des opposants (potentiels) pour tenter de les obtenir, nous commencerons par examiner les types d'attaques dont peuvent disposer ces opposants.

Nous verrons ensuite les principes de base qui en découlent et les concepts d'entropie et de sécurité parfaite.

2.2.1. La cryptanalyse, les types d'attaques

Une des questions essentielles est de connaître la difficulté à déchiffrer le message sans connaître la clé.
Dans les cas repris plus haut, une table reprenant les statistiques d'occurrence de chacune des lettres dans le langage du message sera d'un grand secours au cryptanalyste et permettra de réduire de manière très sensible la complexité de la cryptanalyse et de « craquer » les systèmes décrits avec une rapidité déconcertante[3].

Les attaques de l'opposant sont multiples :

L'opposant peut recevoir uniquement les textes chiffrés, et c'est à partir de ces textes qu'il essaie de découvrir la clé (et, bien entendu, par voie de conséquence, le message, objectif principal) .

---

[3] Voir par exemple la section cryptanalyse du chiffrement de Blaise de Vigenere de Stinson : Douglas R. Stinson, *Cryptography Theory and Practice*, Boca Raton, CRC Press, 1995, pages 31-36.

Nous parlerons dans ce cas d'une **attaque à textes chiffrés**. C'est l'attaque la plus simple à réaliser, il suffit de capturer les messages sur le canal de transmission.

L'opposant peut également mettre en place une **attaque à mots connus ;** il connaît les mots du texte en clair et leur correspondant chiffré. Cette supposition n'est pas dénuée de tout sens ; il est raisonnable de penser que des parties de message sont connues. Il peut s'agir de l'entête, de formules consacrées ou encore de début de fichier informatique standard, telle la première ligne d'une base de données Access, le protocole utilisé ou des données similaires.

Le cryptanalyste peut aussi choisir les textes en clair et recevoir le correspondant chiffré ; cette attaque peut être plus efficace car les textes en clair peuvent être choisis de telle sorte que des informations sur la clé puissent être obtenues plus rapidement. Cette attaque est appelée l'**attaque à mots clairs choisis**. Pour effectuer ce genre d'attaque, nous estimerons que l'attaquant a accès au système de chiffrement sans pouvoir en obtenir la clé.

Afin de réaliser des estimations, quelques hypothèses et concepts supplémentaires de travail doivent être envisagées, ceux-ci sont décrits dans les sections qui suivent.

2.2.2. Principe de Kerckoff

Une des hypothèses est le principe de Kerckoff selon lequel il faut considérer la valeur de l'algorithme en estimant que l'opposant a la connaissance du système cryptographique utilisé, de la méthode, de la longueur de la clé ou du mot code.
La seule connaissance manquante est la clé utilisée.

2.2.3. Systèmes sûrs ou sécurité parfaite

Un système est réputé sûr s'il peut être prouvé qu'en utilisant la meilleure attaque connue, le système ne peut être craqué en un temps raisonnable. Il ne s'agit bien sûr pas d'une preuve absolue mais d'une situation similaire à celle de prouver qu'un problème est NP-complet.

La sécurité parfaite, quant à elle, a été définie par Claude Shannon dans un article[4] en 1949. La théorie de Shannon a influencé toute la cryptographie contemporaine. Les paragraphes qui suivent vont tenter de résumer les principaux concepts de cette théorie.

L'idée de base est la suivante : un système est parfaitement sûr si l'on peut prouver qu'aucune information sur le texte clair ne peut être obtenue en analysant le texte chiffré.

---

[4] Claude Shannon, *Communication Theory of Secrecy System*, Bell Technical Journal, 1949

Une des réalisations les plus connues d'un système parfaitement sûr est le « one-time pad[5] ». Il s'agit ici d'utiliser une clé aléatoire aussi longue que le message à chiffrer et d'additionner la clé au texte clair.Cette clé ne sert qu'une seule fois.

Ce système fut utilisé pendant de longues années mais la transmission sécurisée de la clé pose autant de problème que la transmission du message qu'elle doit chiffrer. De plus ce système ne convient aucunement aux besoins modernes du chiffrement.

2.2.4. Entropie et compression

Puisque la même clé devra être utilisée pour chiffrer plusieurs messages, il convient de s'intéresser à la redondance du message et à l'unicité de son chiffrement. Ces deux concepts que nous définirons ci-dessous ont une influence sur la difficulté qu'aura l'opposant à découvrir le message qui se cache derrière le texte chiffré.

2.2.4.1. Définition de l'entropie

L'entropie définit le nombre optimal de bits nécessaires pour encoder l'information contenue dans un message ou un langage. Il est clair que dans le langage, toutes les combinaisons ne sont pas possibles et qu'une certaine redondance subsiste.

Plus formellement, l'entropie est définie comme suit :

$$H(X) = -\sum_{i=1}^{n} p(X = x_i).\log_2 p(X = x_i)$$

H (X) est l'entropie du langage X
$p(X = x_i)$ est la probabilité de retrouver la valeur x_i dans le langage X.

Le logarithme en base 2 est utilisé lorsqu'il s'agit d'encoder le langage binaire. Plus le nombre de bits utilisés est supérieur à l'entropie, plus le langage est redondant et facile à cryptanalyser !

2.2.4.2. Exemple 1 : Codage simple

Soit un alphabet Â constitué de 5 lettres associées à leur probabilité de survenance :

```
A : 0,5
B : 0,2
C : 0,15
D : 0,1
E : 0,05
```

Cet alphabet pourrait être codé simplement de la manière suivante en utilisant 3 bits :

---

[5] Le one-time pad fut inventé par Gilbert Vernam en 1917 ; ce n'est qu'en 1949 que la preuve de son « incassabilité » fut démontrée.

```
A : 000
B : 001
C : 010
D : 011
E : 100
```

La valeur de l'entropie d'un tel système est la suivante :

$H(\hat{A}) = -0,5 \log_2 0,5 - 0,2 \log_2 0,2 - 0,15 \log_2 0,15 - 0,1 \log_2 0,1 - 0,05 \log_2 0,05$

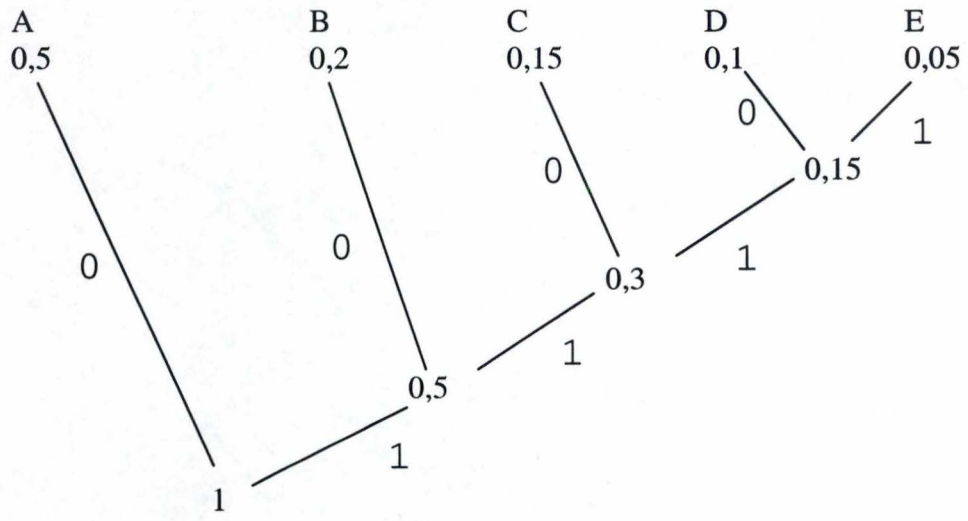
$H(\hat{A}) = 1,92$

Il faudrait idéalement 1,92 bit pour coder l'alphabet. En utilisant un code à 3 bits, le système est redondant par 1,08 bits et nous nous apercevons que toutes les possibilités ne sont pas utilisées.

2.2.4.3. Exemple 2 : le codage de Huffman

Le codage de Huffman est un système qui tient compte de la probabilité associée à l'élément de l'alphabet $\hat{A}$ et qui tente de s'approcher du codage idéal défini plus haut. C'est un système basé sur une arborescence : à chaque itération, les éléments de plus basses probabilités sont associés pour donner un élément dont la probabilité est la somme des probabilités et ainsi de suite. Le codage est ensuite opéré, l'élément le plus petit reçoit la valeur 1, l'autre la valeur 0.

Notre exemple prendra la forme suivante



Le codage de Huffmann donne le résultat suivant :

```
A : 0
B : 10
C : 110
D : 1110
E : 1111
```

et la longueur moyenne du codage l(x) s'approche de l'entropie :

$l(x) = 4 . 0,05 + 4 . 0,1 + 3 . 0,15 + 2 . 0,2 + 1 . 0,5$

$l(x) = 1,95$ (contre 1,92)[6]

Remarquons également que, malgré un codage employant un nombre de bits différents selon l'information, le codage de Huffmann est établi de telle manière qu'aucune ambiguïté ne subsiste quant à la lecture de cette dernière[7].

Il ne faut donc pas s'étonner que la plupart des réalisations cryptographiques, comme PGP[8], par exemple, utilisent un programme de **compression** avant de chiffrer les messages. Il diminue de cette façon la redondance qui y est associée.
L'anglais, par exemple, exprimé en code ASCII est redondant par 6,8 bits sur 8, soit à plus de ¾ redondant !

2.2.5. La distance d'unicité

Elle définit la longueur nécessaire du texte chiffré pour qu'une seule clé donne raisonnablement une solution dans le langage déterminé.
La distance d'unicité s'exprime par :

$$\boxed{U = H(k) / R}$$

H(k) étant la mesure de l'entropie du système Crypto
R la redondance du langage.

Plus la distance d'unicité est grande, meilleur est le système cryptographique. Prenons un algorithme possédant une clé de 128 bits et un message codé en anglais, nous obtiendrons une distance d'unicité de 128/6,8 soit 18,8 caractères ASCII soit 150 bits. Cela veut dire qu'il est possible qu'un texte de moins de 150 bits chiffré avec ce système possède plusieurs déchiffrements plausibles et donc est plus difficile à cryptanalyser car l'attaquant ne peut pas choisir le bon déchiffrement ; toutes les solutions découvertes sont équiprobables.

---

[6] On peut démontrer que la longueur moyenne du codage de Huffmann est toujours comprise entre H(x) et H(x) + 1

[7] Ce codage est utilisé dans de nombreux logiciels de compression comme WinZip par exemple.

[8] PGP : Pretty Good Privacy est un logiciel de chiffrement conçu par Philip Zimmerman, il est disponible gratuitement sur www.pgpi.com .

Un système Crypto possédant une distance d'unicité infinie est donc un système à sécurité parfaite (le 'one time pad' répond à cette condition puisque l'entropie du système est infinie).

2.2.5.1. Confusion et diffusion

Dans son article, Shannon met en évidence deux principes qui doivent guider les créateurs de systèmes de chiffrement, la confusion et la diffusion.

La diffusion est la dispersion de l'information du texte en clair dans la totalité du texte chiffré. Cette dispersion camoufle la structure du texte clair et rend la cryptanalyse plus difficile.

Par la confusion, Shannon entend l'utilisation de transformations qui rendent plus compliquée la détermination des statistiques du texte en clair.

Par l'expression de ces deux concepts clés, les principes de la cryptographie classique ont été conservés : la transposition et, plus particulièrement la permutation, réalisent la diffusion de l'information dans le message. La substitution réalise la confusion en gommant les relations avec le texte clair. Ces deux mécanismes utilisés ensemble sont encore le cœur de la plupart des chiffrements modernes que nous évoquerons plus loin.

Reprenons notre exemple et opérons successivement la substitution et la permutation définies plus haut, nous obtiendrons rapidement une solution très solide :

```
LESSANGLOTSLONGSDESVIOLONS
dabbtpidjwfdjpibkabohjdjpbda
bpatidbwjjdpdfkobbhiajddbajp
```

3. LE DATA ENCRYPTION STANDARD (DES)

3.1. Introduction

En 1972 le bureau national américain des standards (NBS), dépendant du département du commerce, initia un programme pour la protection des données informatisées. Ce bureau souligna l'importance grandissante du besoin en sécurité informatique pour le gouvernement américain.
L'appel d'offre fut lancé le 15 mai 1973 pour l'acquisition d'un algorithme Crypto standard.
Les principaux critères spécifiés par le NBS pour l'algorithme furent les suivants, cet algorithme devait être :

- D'un niveau de sécurité élevé, c'est-à-dire solide 'cryptographiquement'
- Gratuit
- Public (selon le principe de Kerckoff défini plus haut toutes les spécifications devaient être expliquées et publiées)
- Rapide (en nombre d'opérations nécessaires au chiffrement, à l'initialisation de la clé, au déchiffrement...)
- Exportable (selon la loi qui, à l'époque, interdisait l'exportation de certains algorithmes cryptographiques).

A l'issue de cet appel, le NBS ne reçut aucune soumission satisfaisant au cahier des charges.

Le 27 août 1994, le NBS devenu entre-temps le NIST « National Institute of Standards and Technology) lança un deuxième appel et l'algorithme LUCIFER développé par une équipe d'IBM vers la fin des années 60 fut le candidat le plus valable.

IBM abandonna alors ses droits sur le système qui fut modifié (spécialement les S-boxes que nous détaillerons plus loin) avec le concours de la NSA et rebaptisé DES pour Data Encryption Standard.Finalement, le DES fut adopté le 23 novembre 1976 et publié officiellement le 15 janvier 1977.

Diffie et Hellman, certainement deux parmi les plus grands cryptographes contemporains, soulignèrent à l'époque la valeur relative de l'algorithme proposé : « whit Diffie and I have become concerned that the proposed data encryption standard, while probably secure against commercial assault, may be extremely vulnerable to attack by an intelligence organization [9]».

Une révision quinquennale du système fut décidée et le DES fut reconduit jusque en 1997.

---

[9] G. J. Simmons, *Contemporary Cryptology, The Science of Information Integrity*, p.48, IEEE press, Piscataway, NJ, 1992

3.2. Description du DES

3.2.1. Schéma général

Le DES est un système de chiffrement par blocs. L'algorithme utilise une clé de 56 bits pour chiffrer des blocs d'une longueur de 64 bits et ainsi obtenir un nouveau bloc chiffré de 64 bits. (Voir Fig.3.1)
Sur base d'un texte en clair, l'algorithme effectue les opérations suivantes :

a. Les bits subissent une permutation initiale (IP).

b. Ils sont séparés en deux blocs de 32 bits (Lo, Ro)

c. 16 itérations ou round du schéma de la fig. 3.1 sont ensuite opérées :
avec

$L_i = R_{i-1}$
$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$
K_i = 48 bits de la clé K (décrite plus loin selon le Key Schedule)

d. Les deux blocs sont rassemblés et subissent une permutation finale (IP^{-1}), inverse de la permutation initiale.

3.2.2. La fonction F

La fonction F reçoit deux paramètres en entrée [$F(R_{i-1}, K_i)$].
Les 32 bits de R_{i-1} subissent une expansion E au terme de laquelle R_{i-1} sera de la même taille que K_i (48 bits).

Cette expansion $E(R_{i-1})$ est une simple permutation dans laquelle certains bits apparaissent deux fois[10].

Le résultat sera ensuite additionné à K_i et les 48 bits formeront 8 groupes de 6 bits qui se présenteront à l'entrée des S-Boxes (S) susmentionnées.

Les tables de substitution S-Boxes sont compressives et le résultat donnera 32 bits qui après une nouvelle permutation s'additionneront à L_{i-1} pour donner R_i. En annexe A se trouvent les données des différentes permutations, les S-boxes et les tables d'expansion.

$R_i = F(R_{i-1}, K_i)$ xor L_{i-1}
$F(R_{i-1}, K_i) = S[E(R_{i-1})$ xor $K_i]$

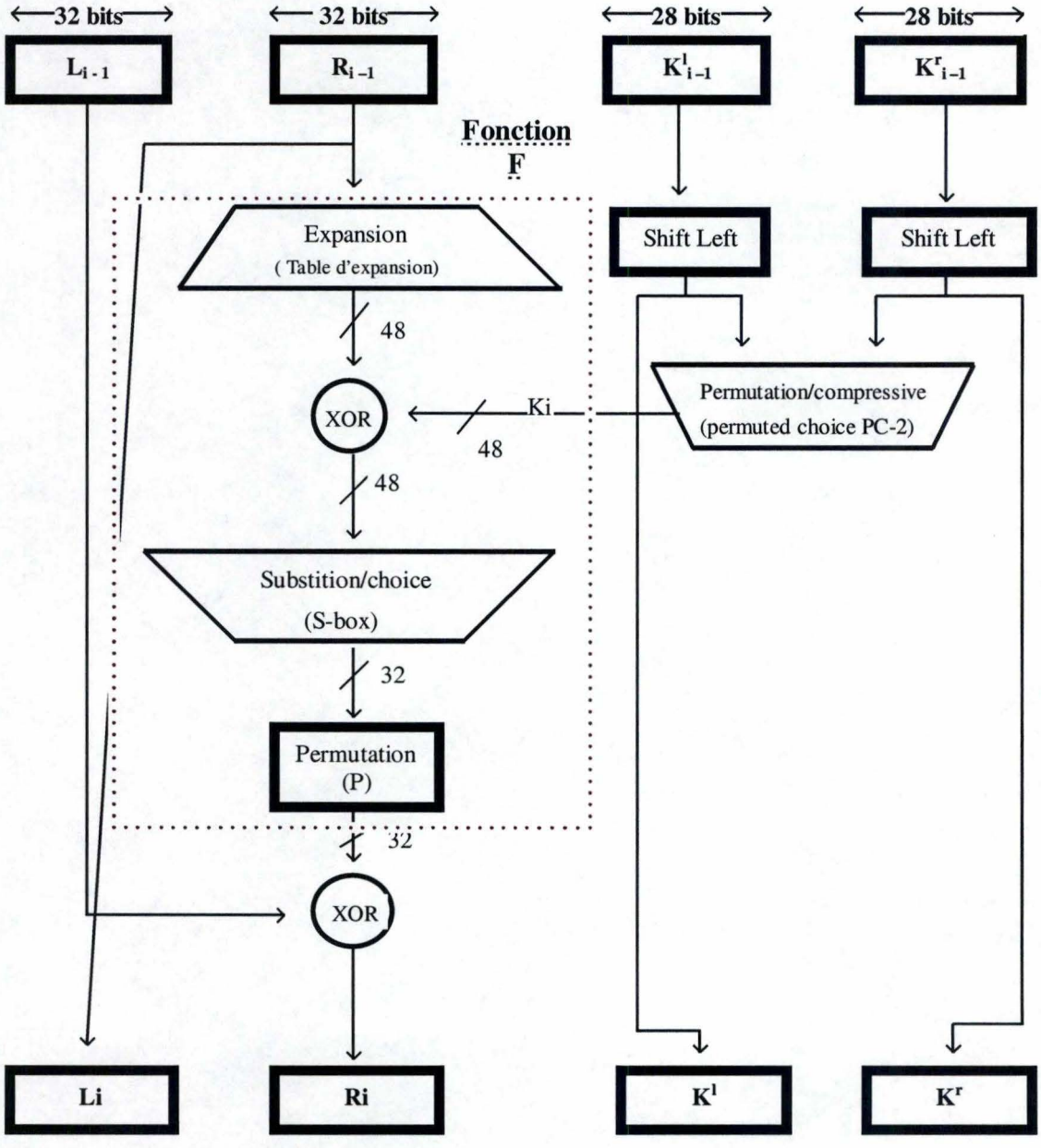
3.2.3. La clé du DES

Les 56 bits significatifs des 64 bits de la clé subissent une permutation initiale appelée « permutated choice 1 » (PC-1). Ils sont divisés en 2 groupes de 28 bits qui opèrent une rotation gauche d'un ou deux bits selon le round (Circular Shift Left) et sont de nouveau permutés de manière compressive(Permutated Choice 2) pour former K_i.

---

[10] Voir tableau de permutation E en annexe A

Fig. 3.1 Schéma d'un round du DES[11]



[11] Les permutations IP et IP^{-1} ne sont pas représentées.

3.2.4. Le déchiffrement du DES

Puisqu'il s'agit d'un chiffrement respectant le schéma de Feistel, le déchiffrement s'effectue de la même manière que le chiffrement à l'exception de la clé dont le processus est à inverser (reversed key scheduling). Avant le déchiffrement les parties gauches et droites doivent être inversées. Ce système simplifie fortement l'implémentation de l'algorithme.

Montrons que cela est vrai en considérant la fonction F comme une boîte noire et en testant sur 2 rounds du DES.

Soient L_0,R_0 respectivement les parties gauches et droites au début du round 1
L_1,R_1 respectivement les parties gauches et droites au début du round 2
L_2,R_2 respectivement les parties gauches et droites au début du round 3
K_1, K_2, K_3, les clés des rounds 1,2 et 3
L_0',R_0',L_1',R_1',L_2',R_2'ces mêmes parties dans l'algorithme de déchiffrement.

- Round 1

  L_0
  R_0

- Round 2

  $L_1 = R_0$
  $R_1 = F(R_0, K_1) \oplus L_0$

- Round 3

  $L_2 = R_1$
  $R_2 = F(R_1, K_2) \oplus L_1$

Déchiffrons en utilisant le processus de clé inversé.

- Round 1 (avec K_2)

  $L_0' = F(R_1, K_2) \oplus L_1$
  $R_0' = L_2 = R_1$

- Round 2 (avec K1)

  $L_1' = R_0' = R_1$
  $R_1' = F(R_0', K_2) \oplus L_0' = F(R_1, K_2) \oplus F(R_1, K_2) \oplus L_1 = L_1$

- Round 3

  $L_2' = R_1' = L_1 = R_o$
  $R_2' = F(R_1', K_1) \oplus L_1' = F(R_0, K_1) \oplus F(R_0, K_1) \oplus L_0$

3.3. Principales caractéristiques du DES

3.3.1. Les S-Boxes

Les seuls composants non linéaires du DES sont les S-Boxes, elles sont d'une importance vitale dans la sécurité du système Crypto. Ce sont elles qui opèrent la confusion. Leurs principales propriétés en ont été énumérées par l'Agence Nationale de Sécurité américaine (la NSA) :

(1) Chaque ligne de la S-Box est une permutation contenant les 16 possibilités de sortie.
(2) Le changement de 1 bit à l'entrée d'une des 8 S-Boxes modifie la sortie d'au moins 2 bits.
(3)Aucune S-Box n'est une fonction linéaire de l'entrée.

3.3.2. Le nombre de rounds

Il s'agit ici d'un compromis sécurité/efficacité : plus le nombre de rounds est élevé, plus la cryptanalyse est difficile. Comme nous le verrons plus loin, le nombre de 16 rounds est judicieux car il rend l'attaque brutale presque aussi efficace que l'attaque différentielle[12] (de l'ordre de 2^{55} mots connus).

3.3.3. Les permutations

Toutes les permutations dont il est question plus avant assurent une très grande diffusion à la sortie de chaque S-Box, les 4 bits de sortie de chaque S-Box affectent, au round suivant, l'entrée de 6 S-Boxes (par un et un seul bit chacune !)

3.3.4. La clé

Comme nous l'avons vu, la clé est composée de 56 bits (choisis idéalement de manière aléatoire). Certaines faiblesses ont été découvertes en étudiant les clés du DES.

Des **clés dites faibles** (fig. 3.2): comme nous l'avons décrit plus haut, les clés sont séparées en 2 groupes et sont décalées selon le round, il en découle que des clés composées uniquement de bits à 1 ou à O après la permutation initiale (PC-1) sont faibles car la même clé est utilisée à chaque round et donne en résultat le même texte que le texte en clair.

Fig. 3.2 : Clés faibles du DES

00000000000000000000000000000000	00000000000000000000000000000000
00000000000000000000000000000000	11111111111111111111111111111111
11111111111111111111111111111111	00000000000000000000000000000000
11111111111111111111111111111111	11111111111111111111111111111111

[12] Ce qui présume que l'attaque différentielle était connue par la NSA en 77 alors qu'elle fut « découverte » par Biham et Shamir en 1993 !

Des **clés semi-faibles**, sont des clés engendrant des sous-clés utilisées plusieurs fois dans l'algorithme.

Des **clés complémentaires** telles que le chiffrement du complément d'un texte avec cette clé donne le complément du texte chiffré avec la clé originale :

Soit k la clé et ¬k, son complément, x, le message,
Si $S_k(x) = y$
Alors $S_{\neg k}(\neg x) = \neg y$

3.4. L'attaque exhaustive ou brutale

De la longueur de la clé est déduite l'ampleur de l'attaque exhaustive appelée aussi attaque brutale[13], avec 2^{56} clés possibles, la clé est à priori découverte après, en moyenne, 2^{55} tentatives soit ~3 10^{16} recherches et compte tenu des faiblesses dans les clés, la recherche peut s'arrêter après un peu moins de 2^{55} clés.

3.5. L'attaque différentielle

L'attaque différentielle a été découverte par Biham et Shamir en 1993[14]. Cette attaque reste théorique sur une version réelle du DES en 16 rounds.
Elle n'en reste pas moins révolutionnaire et elle a été prise en compte lors de la conception du RIJNDAEL et des autres algorithmes candidats AES.

3.5.1. Idée générale

L'attaque considère chaque round du DES en utilisant des paires de textes en clair dont la différence (XOR) est connue. (soit $L_o'R_o' = L_oR_o$ XOR $L_o^* R_o^*$) .

La permutation initiale peut être négligée puisqu'elle n'ajoute aucune sécurité au système[15].

Considérons 2 groupes de six bits à l'entrée d'une S-Box (B_j et B_j^*), la différence en entrée s'écrira B_j' ($=B_j$ XOR B_j^*) et la différence en sortie $S_j'(B_j)$ [= $S(B_j^*)$ XOR $S(B_j)$].

Prenons par exemple les 32 paires possibles donnant un B_j' donné et évaluons la sortie, nous obtiendrons une distribution non uniforme des 16 sorties possibles.

Voici un exemple avec une différence (B_j')de 010000 dans la première S-Box (Fig. 3.3), nous obtiendrons pour la paire suivante:

---

[13] La littérature sur le sujet utilise généralement les termes « Exhaustive search » ou « Brute attack ».
[14] E. Biham et A Shamir, *Differential cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 1993
[15] Peu d'explications sont, en fait, fournies sur la cause de son existence.

$B_j = 001000$
$B_j^* = 011000$
} $B_j' = 010000$ (16)

$S(B_j) = 0010$
$S(B_j^*) = 0101$
} $S(B_j') = 0111$ (7)

.

Fig. 3.3 : S-Box 1 (S1)

Les 2 bits extérieurs déterminent la ligne tandis que les 4 bits intérieurs déterminent la colonne

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

En analysant toutes les paires possibles nous obtiendrons le tableau de distribution[16] suivant :

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
0	*2*	*0*	*0*	*0*	*0*	*4*	*16*	*2*	*4*	*6*	*12*	*2*	*4*	*8*	*4*

Le 7 a une probabilité d'apparition de ¼.

Observons également, et c'est ici l'idée maîtresse de l'attaque, que B_j' ne dépend pas de la clé. Tandis que la différence en sortie en dépend.

En effet

B_j XOR B_j^* devient avant le passage dans la S-Box
$(B_j$ XOR $K_j)$ XOR $(B_j^*$ XOR $K_j) = B_j$ XOR B_j^*

[16] La première ligne représente la différence obtenue $S(B_j')$ sous forme décimale, la deuxième ligne le nombre d'occurrences de cette différence pour les 32 paires possibles ayant un $B_j' = 16$

3.5.2. La 'caractéristique'

La caractéristique est associée à la probabilité qu'une paire de texte en clair, prise au hasard mais dont la différence est connue, donne une différence en sortie spécifiée.

Sur 1 round, par exemple, une différence de 16 (010000) dans la première S-Box et de 0 pour les autres donnera une différence de 7 en sortie avec une probabilité de 16/64 soit ¼.

Trivialement une différence de 0 donnera une différence en sortie de 0 avec une probabilité de 1. (cette propriété est utilisée pour la recherche de certaines caractéristiques).

En effet si $B_j' = 0 \rightarrow S(B_j') = 0$ [17]

Les paires de textes qui suivent la caractéristique sont appelées les bonnes paires, les autres, les mauvaises paires.

Une caractéristique peut être établie sur n-rounds en multipliant les probabilités établies pour chacun des rounds.

Sur un round, nous avons vu qu'il est possible de trouver une caractéristique de probabilité ¼, elle sera également de ¼ sur deux rounds en choisissant pour la partie gauche une différence de 0 et un contenu égal à la sortie de la fonction F de la partie droite puisque le DES ne chiffre que la partie droite de l'information et inverse les deux parties au round suivant.

Le problème est de trouver une caractéristique qui, après plusieurs rounds, présente encore des probabilités intéressantes.

3.5.3. La caractéristique itérative

Une caractéristique peut également être itérative (Fig. 3.4) si une partie'[18] est égale à 0 et l'autre partie' est choisie de telle sorte qu'à la fin du chiffrement, la même situation se présente avec la partie gauche et droite inversées.

La caractéristique se propagera ainsi à travers les rounds du DES avec une probabilité multipliée.
La meilleure caractéristique itérative trouvée pour le DES a une probabilité de 1/234 (par round).

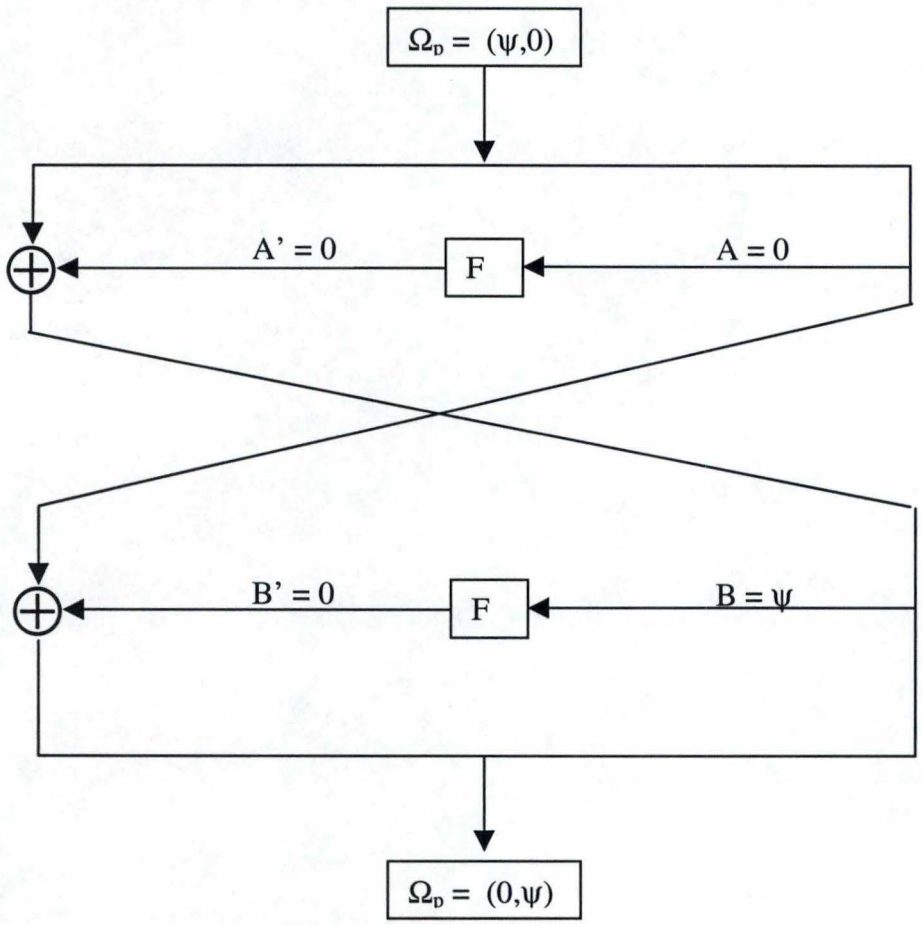
La difficulté est de trouver la caractéristique idéale, c'est-à-dire celle possédant la plus grande probabilité d'occurrence.

---

[17] Il s'agit du même texte !
[18] Notons « partie' », l'opération XOR entre 2 parties.

Biham et Shamir utilisèrent pour choisir la caractéristique citée plus haut une heuristique qu'ils estimaient idéale sans toutefois pouvoir le prouver[19]. Leur meilleure solution nécessitait $2^{55.1}$ mots connus, très légèrement moins intéressante qu'une attaque brutale ou 2^{47} mots choisis.

Fig. 3.4 Schéma d'une caractéristique itérative



[19] E. Biham et A Shamir, *Differential cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 1993, page 28, en fait, il fut prouvé en 1995 que cette caractéristique (B_j=19 60 00 00 00 00 00 00_x) était idéale.

3.5.4. La découverte de la clé

En analysant les 'bonnes' paires, c'est-à-dire les paires qui suivent la caractéristique, la clé suggérée sera plus fréquemment la clé réelle tandis que les 'mauvaises' paires suggéreront toutes les clés possibles de manière uniforme. La bonne clé sera suggérée avec la même probabilité que la caractéristique. Pour découvrir la clé, il suffit donc de compter le nombre d'occurrence de chaque clé suggérée et de prendre la clé la plus souvent suggérée.

En annexe B se trouve un exemple complet d'une cryptanalyse différentielle sur un DES réduit à 6-rounds et utilisant 120 paires de textes clairs.

3.6. La Fin du DES

De 1977 à 1993, le DES fut le seul candidat sérieux à sa réélection ; malheureusement, les critères de design furent très longtemps classifiés confidentiels par la NSA qui ne les révéla qu'au compte-gouttes.

Des rumeurs de « trapdoors » commencèrent à circuler ; d'autant plus que respectivement en 91 et en 93, furent découvertes les attaques différentielles et linéaires. En 1993, le DES fut réélu.

Le 3-DES[20] fut, en fait, standardisé mais l'algorithme n'était pas nouveau, il était lent, chiffrait toujours des blocs de 64 bits et, en fait, convenait peu aux processeurs modernes.

NIST ne manqua d'ailleurs pas de mentionner lors de l'annonce du résultat : « At the next review (1998), the algorithm specified in this standard will be over twenty years old. NIST will consider alternatives wich offer a higher level of security. One of these alternatives may be proposed as a replacement standard at 1998 review ». En 1998, l'EFF construit la machine capable de casser le DES en ... Le glas du DES avait sonné.

Nous allons voir au cours du chapitre qui suit en quoi le nouveau standard comble les lacunes du DES et correspond mieux aux 'attentes'.

[20] Une solution du DES améliorée par le surchiffrement (3 applications du DES)

4. LE RIJNDAEL

4.1. Introduction

L'algorithme RIJNDAEL a été créé par Joan Daemen et Vincent Rijmen. Il fut présenté comme candidat nouveau standard à l'AES (Advanced Encryption Standard)

4.2. La sélection du Rijndael

4.2.1. Le cahier des charges

Le 2 septembre 1997, le NIST initia à nouveau un processus de sélection d'un algorithme. Le « CSRC »[21] spécifie toutes les conditions à remplir par les candidats.

Cette fois, la transparence était de rigueur. Les candidats devaient soumettre au NIST pour le 15 juin 98 un dossier comprenant les éléments suivants.

- Une spécification complète de l'algorithme (méthode de chiffrement, nombre de rounds,...)
- Une estimation la plus documentée possible de l'efficacité de l'algorithme. Le nombre de cycles nécessaires au chiffrement, au déchiffrement, à la mise à clé, au compromis vitesse et mémoire...
- Une estimation la plus complète possible de la solidité cryptographique de l'algorithme en terme de résultats à des tests connus (Known Answer Test, Monte Carlo Test), de résistance aux attaques connues, de restrictions dans le choix des clés (clés faibles, équivalentes, complémentaires..) et de non-existence de 'trapdoors'.
- Une implémentation optimisée en C et en Java.

L'algorithme devait être symétrique à clés secrètes et pouvoir chiffrer des blocs de 128 bits avec une clé de 128, 192 et 256 bits.

Le 20 août 1998, 15 candidats de 12 pays différents étaient retenus pour le premier round de sélection, les commentaires publics étaient les bienvenus. Le second tour de sélection commença en août 98 avec les 5 candidats retenus. Les algorithmes sélectionnés étaient les suivants :

- MARS proposé par C. Burwick de International Business Marchine Corporation (IBM).
- RC6™ conçu par la RSA Laboratories du célèbre R. Rivest.
- RIJNDAEL soumis à l'AES par Joan Daemen de Proton World International et Vincent Rijmen de la Katholieke Universiteit van Leuven (KUL).

[21] CSRC : Computer Security Division Information Technology Laboratory of National Institute of Standards and Technology, http://csrc.nist.gov/encryption/aes/

- SERPENT mis au point par Ross Anderson de Université de Cambridge, par Eli Biham, le cryptographe réputé de Technion et Lars Knusdsen de l'Université de Californie San Diego
- TWOFISH conçu par l'équipe composée de Bruce Schneier, John Kelsey, et Niels Ferguson de Conterpane Internet Security Inc, de Doug Whiting (Hi/fn Inc), David Wagner de l'Université de Californie Berkeley et par Chris Hall de l'Université de Princeton

4.2.2. Les critères d'évaluation

Les critères d'évaluation annoncés par le NIST sont, par ordre d'importance, la sécurité, le coût et les autres caractéristiques des l'algorithme.
Nous décrirons tout au long du chapitre l'algorithme d'une façon relativement détaillée et exprimerons aux chapitres suivants la signification de ces critères afin de montrer comment l'algorithme RIJNDAEL remplit ces critères d'évaluation en les comparant au DES ou éventuellement aux autres candidats finalistes.

4.3. Description

Le RIJNDAEL est un chiffrement par blocs de 128, 192 et 256 bits avec une clé de 128, 192 ou 256 bits en 10, 12 ou 14 rounds. Le tableau ci-après reprend le nombre de rounds spécifiés en fonction de ces deux variables.

Tableau 4.1 : le nombre de round du RIJNDAEL

Nb Rounds	Bloc de 128	Bloc de 192	Bloc de 256
Clé de 128	10	12	14
Clé de 192	12	12	14
Clé de 256	14	14	14

Le RIJNDAEL considère l'information par byte, chaque byte est un élément du champ gallois fini $GF(2^8)$ et représente un polynôme binaire.

(1001 1011) correspond à (x7 + x4 + x3 + x + 1)

L'addition correspond à un XOR et la multiplication à une multiplication polynomiale modulo un polynôme irréductible donné ($m(x) = x^8 + x^4 + x^3 + x^1 + 1$) que nous développerons plus loin.

4.3.1. Etapes de l'algorithme

Nous allons citer les étapes de l'algorithme et pour chacune d'entre elles, spécifier les opérations exécutées.

L'algorithme s'exécute en 3 étapes :

- Un round initial d'addition de clé ('AddRoundKey')
- n-1 rounds divisés en quatre étapes :
  - la substitution appelée 'ByteSub'
  - le décalage des lignes 'ShiftRow'
  - la diffusion en colonnes 'MixColumn'
  - l'addition de la clé 'AddRoundKey'.
- Un dernier round égal aux rounds précédents à l'exception de la diffusion des colonnes.

4.3.1.1. La substitution

La substitution s'effectue par byte (un byte est remplacé par un autre) et s'opère en deux étapes. Tout d'abord il convient de prendre l'inverse de chaque byte (le byte 0000 0000 étant remplacé par lui-même) et ensuite de substituer le résultat par une valeur déterminée par le vecteur Y(Fig. 4.1) qui, en fait, correspond à la transformation affine suivante :

Fig. 4.1 : **Transformation de substitution**

$$\begin{vmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{vmatrix} \begin{vmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{vmatrix} + \begin{vmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{vmatrix}$$

4.3.1.1.1. La recherche de l'inverse

Pour la recherche de l'inverse du byte dans le champ fini, l'algorithme étendu d'Euclide est utilisé. Cette méthode dont le pseudo-code est décrit plus bas est destinée à chercher le PGCD de 2 nombres entiers. Elle peut être appliquée au champ fini de RIJNDAEL :

Soit m(x), le polynôme irréductible du RIJNDAEL : $x^8 + x^4 + x^3 + x^1 + 1$

soient a(x), b(x) et c(x) des éléments du champ fini

si nous avons $b(x)a(x) + m(x)c(x) = 1$
$b(x)\ a(x) = 1 + m(x)\ c(x)$
$b(x)\ a(x) \bmod m(x) = 1 + m(x)\ c(x) \bmod m(x)$

d'où $a(x) \bullet b(x) \bmod m(x) = 1$

alors l'inverse peut être trouvé par :
$b^{-1}(x) = a(x) \bmod m(x)$

Le pseudo-code de l'algorithme d'Euclide est le suivant
Soit a, b ∈ N ; ∃ des entiers u et v tels que ua + vb = PGCD (a , b)

Initialisation avec b>a

$s_0 = b$, $s_1 = a$,
$u_0 = 0$, $u_1 = b$,
$v_0 = 1$, $v_1 = 0$ et $i = 1$

Tant que $S_n > 0$,

$i = i + 1$
$s_i = s_{i-2} - q_i\ s_{i-1}$ (en fait le reste de la division de s_{n-2} par s_{n-1})

$u_i = q_i\ u_{i-1} + u_{i-2}$
$v_i = q_i\ v_{i-1} + v_{i-2}$

4.3.1.1.2. Exemple

Recherchons l'inverse de **$x^5 + 1$ ('21')**

Nous avons $a = x^5 + 1$
$b = x^8 + x^4 + x^3 + x + 1$, le polynôme irréductible

Initialisation

Boucle
$i = 2$
$s_2 = x^4 + x + 1$ venant du reste de la division de b par a
$u_2 = x^3 . 1$
$v_2 = x^3 . 0 + 1 = 1$
$i = 3$
$s_3 = x^2 + x + 1$
$u_3 = x^3 . x + 1 = x^4 + 1$
$v_3 = x . 1$
$i = 4$
$s_4 = x^3 + x^2 + x + 1$
$u_4 = x^6 + x^3 + x^2$

$v_4 = x^3 + 1$

i = 5

$s_5 = x^2 + x + 1$

$u_5 = x^4 + 1$

$v_5 = x$

i = 6

$s_6 = \mathbf{1}$

$u_6 = x^6 + x^5 + x^3 + x^2 + x$

$v_6 = x^3 + x^2 + 1$

i = 7

$s_7 = \mathbf{0}$

Fin de boucle

L'inverse de $(x^5 + 1)$ est $u_6 = x^6 + x^5 + x^3 + x^2 + x$ (l'inverse de '21' est '6E')
et

En effet
$(x^6 + x^5 + x^3 + x^2 + x) \bullet (x^5 + 1) = \quad x^{11} + x^{10} + x^8 + x^7 + x^6 + x^5 + x^3 + x^2 + x$

et le modulo est bien nul et

$x^{11} + x^{10} + x^8 + x^7 + x^6 + x^5 + x^3 + x^2 + x$ modulo $x^8 + x^4 + x^3 + x^1 + 1 = \mathbf{1}$

4.3.1.2. Le décalage des lignes

En arrangeant les bytes en un rectangle de 4 lignes, ils sont décalés en lignes vers la gauche de respectivement 0,1,2 et 3 positions ou de 0,1,3,4 positions s'il s'agit d'un bloc de 256 bits. Le tableau suivant est un exemple reprenant 16 bytes.

Fig.4.2 : Décalage en ligne

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

→

1	5	9	13
6	10	14	2
11	15	3	7
16	4	8	12

4.3.1.3. La diffusion en colonnes

Elle correspond à la multiplication de la colonne dont chaque terme est un polynôme dans le champ fini $GF(2^8)$ par un polynôme c(x) modulo $x^4 + 1$

avec $c(x) = \text{'03'}\ x^3 + \text{'01'}\ x^2 + \text{'01'}\ x + \text{'02'}$

La multiplication (•)

La multiplication, telle que définie dans l'algorithme RIJNDAEL et utilisée dans la diffusion en colonnes, est la multiplication dans le champ fini « $GF(2^8)$ » correspondant à une multiplication polynomiale modulo un polynôme irréductible de degré 8[22].

Ce polynôme appelé m(x), choisi par les auteurs, est le polynôme binaire suivant :

$m(x) = x^8 + x^4 + x^3 + x^1 + 1$ soit `1 0001 1011` `'11B'`

La multiplication par c(x) modulo $x^4 + 1$ ($\otimes$)

Prenons la colonne initiale (avant l'opération) :

soit $i(x) = i_3x^3 + i_2x^2 + i_1x^1 + i_0x^0$
et $c(x) = \text{'03'}\ x^3 + \text{'01'}\ x^2 + \text{'01'}\ x + \text{'02'}$

L'opération $i(x) \otimes c(x)$ donne le résultat $r(x) = r_3x^3 + r_2x^2 + r_1x^1 + r_0x^0$

avec
en x^0 (x^4 ou x^0 modulo $x^4 + 1$) : $r_0 = \text{'02'} \bullet i_0 + \text{'03'} \bullet i_1 + \text{'01'} \bullet i_2 + \text{'01'} \bullet i_3$
en x^1 (x^5 ou x^1 modulo $x^4 + 1$) : $r_1 = \text{'01'} \bullet i_0 + \text{'02'} \bullet i_1 + \text{'03'} \bullet i_2 + \text{'01'} \bullet i_3$
en x^2 (x^6 ou x^2 modulo $x^4 + 1$) : $r_2 = \text{'01'} \bullet i_0 + \text{'01'} \bullet i_1 + \text{'02'} \bullet i_2 + \text{'03'} \bullet i_3$
en x^3 (x^3 modulo $x^4 + 1$) : $r_3 = \text{'03'} \bullet i_0 + \text{'01'} \bullet i_1 + \text{'01'} \bullet i_2 + \text{'02'} \bullet i_3$

Cette opération peut être résumée par la matrice définie en Fig 4.3

Fig. 4.3. Matrice de diffusion en colonne

02	03	01	01
01	02	03	01
01	01	02	03
03	01	01	02

4.3.1.4. L'addition de la clé

A cette étape la clé du round est simplement additionnée (XOR) au résultat. La clé du round est dérivée de la clé principale et a, bien entendu, la même longueur que le bloc à chiffrer.

4.3.2. Le 'Key Schedule'

La clé est dérivée de la clé principale en 2 étapes : tout d'abord une clé étendue 'expanded key' est produite ; ensuite la clé du round est sélectionnée à l'intérieur de cette clé selon le round.

---

[22] Selon l'AES proposal : Joan Daemen et Vincent Rijmen, *The Rijndael Block Cipher, KULeuven*, 1999, http://www.esat.kuleuven.ac.be/~rijmen/rijndael/

4.3.2.1. L'extension de la clé 'Key expansion'

L'extension de la clé est une fonction dépendante de la longueur de la clé utilisée.

Pour les clés d'une longueur de 128 et 192 bits, l'expansion est donnée sous la forme d'un vecteur contenant les clés de tous les rounds (c'est-à-dire un round initial et les n rounds du RIJNDAEL)

Ce vecteur reçoit tout d'abord la clé originale, ensuite la clé du round suivant est ajoutée par mots de 32 bits W[i] étant égaux à W[i-1] XOR W[i-Nk] avec Nk, le nombre de mots de 32 bits de la clé. Pour les mots multiples de Nk, W[i-1] est décalé cycliquement vers la gauche de 8 bits, il est ensuite substitué à un autre mot de 32 bits en utilisant la S-Box décrite en figure 4.1. et additionné (XOR) à un 'round constant'.

Prenons un exemple avec un bloc de 128 bits et une clé de 128 bits (K[0..3]) et déterminons la valeur du vecteur W en mots de 32 bits. Le vecteur W se composera de 11 * 4 mots de 32bits (le round initial et les 10 rounds du RIJNDAEL pour une telle configuration. Cherchons W[1..44]

W[0..3] = K[0..3]
W[4] = (W[3])' $\oplus$ W[0] $\oplus$ RC[0]
W[5] = W[4] $\oplus$ W[1]
W[6] = W[5] $\oplus$ W[2]
W[7] = W[6] $\oplus$ W[3]
W[8] = (W[7])' $\oplus$ W[4] $\oplus$ RC[1]

avec

(W[3])' = Substitution(CircularShiftLeft$_8$ W[3])
et RC[i] = RC[i-1] • '02'
et RC[0] = ('O1' '00' '00' '00')

et ainsi de suite...

Pour les clés de 256 bits, il s'agit de la même opération que celle décrite plus haut à la différence suivante : pour chaque W[i-4] multiple de Nk une nouvelle substitution est opérée à W[i-1].

4.3.2.2. La sélection de la clé du round 'the round key selection'

La clé du round i est simplement choisie dans le vecteur W et correspond aux mots de W[Nb*i] à W[Nb*(i+1)]

4.3.3. Le déchiffrement du RIJNDAEL

Le système RIJNDAEL n'est pas un schéma de Feistel mais un « substitution-linear transformation Network »[23]. L'inverse de tout le chiffrement ne s'obtient pas en chiffrant le résultat avec la clé inverse (Reverse Key Scheduling) mais en inversant chaque étape.

Toutes les étapes du RIJNDAEL peuvent, en effet, être inversées :

- la substitution s'obtient en appliquant la table inverse (InvByteSub),
- le décalage des lignes en inversant le décalage vers la gauche de 0, 1, 2, (3 ou 4) positions (InvShiftRow),
- la diffusion en colonnes de la même façon en appliquant la même opération (⊗) avec le polynôme inverse (InvMixColumn):

$$c^{-1}(x) = (0B\ x^3 + 0D\ x^2 + 09\ x + 0E)$$

- le 'Key scheduling' reste le même, seule la seléction de la clé s'effectue dans l'ordre inverse[24] (AddRoundKey).

Les propriétés algébriques du chiffre sont telles que le déchiffrement peut être obtenu en gardant le même ordre et en appliquant les opérations inverse décrites plus haut.

Le chiffrement inverse s'obtient donc de la manière suivante :

- Un round initial d'addition de clé ('AddRoundKey' – clé du dernier round)

- n-1 rounds divisés en quatre étapes :

  - la substitution inverse appelée 'InvByteSub'
  - le décalage inverse des lignes 'InvShiftRow'
  - la diffusion inverse en colonnes 'InvMixColumn'
  - l'addition de la clé 'AddRoundKey'.

---

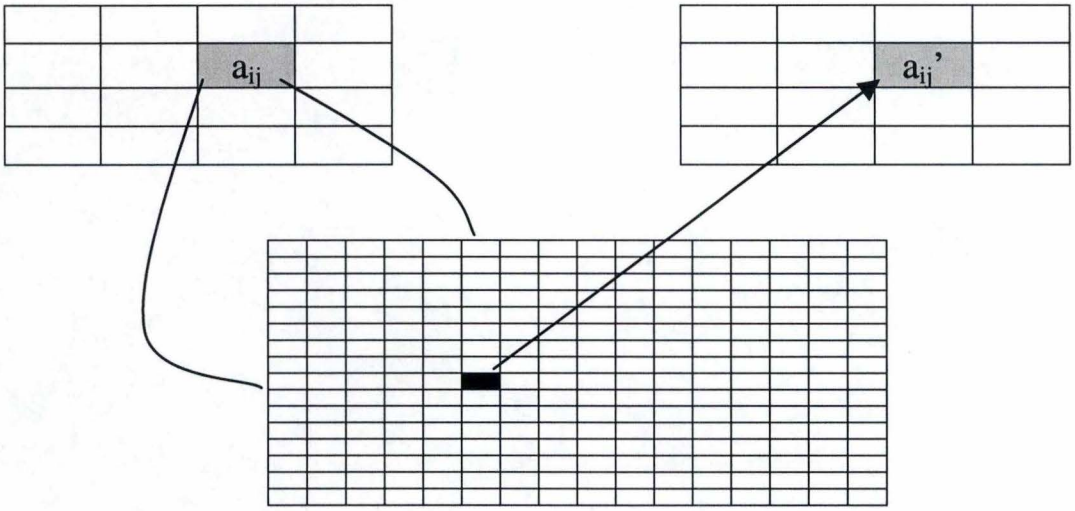
[23] James Nechvatal, Elaine Barker, Lawrence Bassham, William Burr, Morris Dworkin, James Foti, Edward Roback, *Report ont the development of the AES*, NIST, 02 octobre 2000

[24] Il faut bien entendu calculer tout le schedule de la clé (Key Schedule) avant de commencer à déchiffrer le premier bloc.

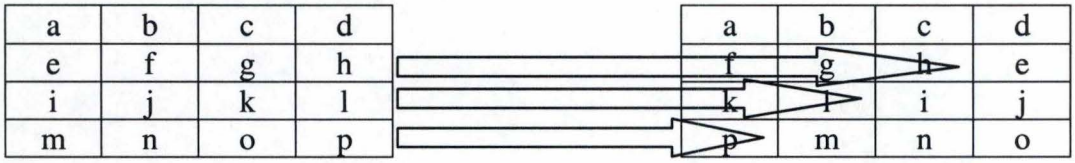
- Un dernier round égal aux rounds précédents à l'exception de la diffusion des colonnes.

4.4. Schéma du RIJNDAEL

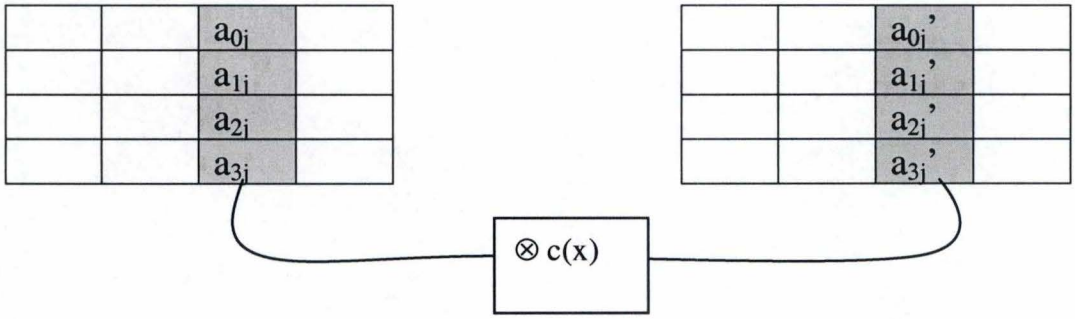
4.4.1. Substitution par byte (ByteSub)



4.4.2. Décalage des lignes (ShiftRow)



4.4.3. Diffusion en colonnes (MixColumn)



4.4.4. L'addition de la clé (AddRoundKey)

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

⊕

k_{00}	k_{01}	k_{02}	k_{03}
k_{10}	k_{11}	k_{12}	k_{13}
k_{20}	k_{21}	k_{22}	k_{23}
k_{30}	k_{31}	k_{32}	k_{33}

=

a'	b'	c'	d'
e'	f'	g'	h'
i'	j'	k'	l'
m'	n'	o'	p'

4.5. Exemple

Au terme de cette brève description, prenons un exemple et faisons le passer dans l'algorithme et ensuite, nous essaierons de comparer le RIJNDAEL avec le DES et éventuellement d'autres algorithmes, candidats à l'AES.

Reprenons les opérations du RIJNDAEL :

Initialement :	Addition de la clé	AddRoundKey
n-1 rounds :	Substitution par byte	ByteSubstitution
	Décalage des lignes	ShiftRow
	Diffusion en colonnes	MixColumn
	Addition de la clé	AddRoundKey
Dernier round :	Substitution par byte	ByteSubstitution
	Décalage des lignes	ShiftRow
	Addition de la clé	AddRoundKey

Prenons une clé de 128 bits et un bloc de 128 bits également, nous devrons alors effectuer 10 rounds et utiliser un vecteur de clé (W[i]) composé de 1408 bits.

Nous avons choisi une clé et un texte particuliers pour montrer à quelle rythme la confusion et la diffusion s'opère. Pour des raisons de clarté, seul le premier round sera expliqué, les résultats seront repris en binaire (tableau de gauche) et en hexadécimal à droite.

Soit la clé suivante :

00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000001

00	00	00	00
00	00	00	00
00	00	00	00
00	00	00	01

et le bloc suivant

00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000010

00	00	00	00
00	00	00	00
00	00	00	00
00	00	00	02

Le Round initial donne après addition de la clé :

00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000011

00	00	00	00
00	00	00	00
00	00	00	00
00	00	00	03

Le premier round :

Substitution : Inverse[25] et multiplication par la matrice de substitution

00000000 → 0110 0011
00000011 → 0111 1011

01100011	01100011	01100011	01100011
01100011	01100011	01100011	01100011
01100011	01100011	01100011	01100011
01100011	01100011	01100011	01111011

63	63	63	63
63	63	63	63
63	63	63	63
63	63	63	7B

Décalage des lignes :

01100011	01100011	01100011	01100011
01100011	01100011	01100011	01100011
01100011	01100011	01100011	01100011
01100011	01100011	01111011	01100011

63	63	63	63
63	63	63	63
63	63	63	63
63	63	7B	63

Diffusion en colonnes

'63' . '01' = '63'
'63' . '02' = 'C6'
'63' . '03' = 'A5'
'7B' . '01' = '7B'
'7B' . '02' = 'F6'
'7B' . '03' = '8D'

devient

les colonnes 1, 2 et 4 deviennent

b = '63'. '02' ⊕ '63'. '03' ⊕ '63'. '01' ⊕ '63'. '01'
= 'C6' ⊕ 'A5'= '63'

La colonne 3 devient

[25] Selon la méthode décrite précédemment (Euclide), l'élément nul est remplacé par lui-même.

b_{03} = '63'. '02' ⊕ '63'. '03' ⊕ '63'. '01' ⊕ '7B'. '01'= '7B'
b_{13} = '63'. '01' ⊕ '63'. '02' ⊕ '63'. '03' ⊕ '7B'. '01'= '7B'
b_{23} = '63'. '01' ⊕ '63'. '01' ⊕ '63'. '02' ⊕ '7B'. '03'= '4B'
b_{33} = '63'. '03' ⊕ '63'. '01' ⊕ '63'. '01' ⊕ '7B'. '02'= '53'

ce qui donne après la diffusion en colonnes

01100011	01100011	01111011	01100011
01100011	01100011	01111011	01100011
01100011	01100011	01001011	01100011
01100011	01100011	01010011	01100011

63	63	7B	63
63	63	7B	63
63	63	4B	63
63	63	53	63

addition de la clé

Voici comment s'étend la clé :

W[0] à W[3] s'étendent pour devenir W[4..7] de la façon suivante :

W[4] = (W[3])' ⊕ W[0] ⊕ RC[0]
W[5] = W[4] ⊕ W[1]
W[6] = W[5] ⊕ W[2]
W[7] = W[6] ⊕ W[3]

avec

(W[3])' = Substitution(CircularShiftLeft$_8$ W[3])
et RC[i] = RC[i-1] • '02'
RC[0] = ('01' '00' '00' '00')

La clé était la suivante

W[0]	W[1]	W[2]	W[3]
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000001

00	00	00	00
00	00	00	00
00	00	00	00
00	00	00	01

Pour **W[4]** les opérations ci-dessous doivent être effectuées :

- Rotation cyclique d'un byte vers la gauche de W[3] qui donne en hexadécimal ['00' '00' '01' '00']
- Substitution du résultat ['63' '63' '7C' '63']
- XOR avec W[0] (sans changement dans ce cas)
- XOR d'une constante de round égale à : ['01' '00' '00' '00'] donne finalement ['62' '63' '7C' '63']

W[5] et **W[6]** sont égaux à W[4] dans notre cas (XOR avec W[1] et W[2])

- **W[7]** = ['62' '63' '7C' '63'] ⊕ ['00' '00' '00' '01'] = ['62' '63' '7C' '62']

La clé prend donc la forme suivante :

W[4]	W[5]	W[6]	W[7]
01100010	01100010	01100010	01100010
01100011	01100011	01100011	01100011
01111100	01111100	01111100	01111100
01100011	01100011	01100011	01100010

62	62	62	62
63	63	63	63
7C	7C	7C	7C
63	63	63	62

ajouté au texte :

01100011	01100011	01111011	01100011
01100011	01100011	01111011	01100011
01100011	01100011	01001011	01100011
01100011	01100011	01010011	01100011

63	63	7B	63
63	63	7B	63
63	63	4B	63
63	63	53	63

Le résultat du premier est le suivant :

00000001	00000001	00011001	00000001
00000000	00000000	00011000	00000000
00011111	00011111	00110111	00011111
00000000	00000000	00110000	00000001

01	01	19	01
00	00	18	00
1F	1F	37	1F
00	00	30	01

5. LE CRITÈRE DE SOLIDITE CRYPTOGRAPHIQUE

Ce critère est certainement un des critères principaux à prendre en considération si l'on veut élire un algorithme pour quelques décennies.

Ce critère évalue la résistance de l'algorithme à la cryptanalyse, la clarté de ses fondements mathématiques et la qualité en termes de confusion et de diffusion (sujet que nous avons déjà abordé au cours du chapitre 2).

5.1. La résistance à l'attaque exhaustive

Dans le cas du RIJNDAEL, cette attaque s'effectue sur la clé ou sur le texte selon le mode choisi et nécessite en moyenne un effort de

- 2^{127} applications du RIJNDAEL ce qui correspond à +/- 10^{38} pour une clé de 128 bits.
- 2^{191} applications du RIJNDAEL ce qui correspond à +/- 10^{57} pour une clé de 192 bits.
- 2^{255} applications du RIJNDAEL ce qui correspond à +/- 10^{76} pour une clé de 256 bits.

Remarquons que dans le DES, la recherche exhaustive ne nécessite que 2^{55} essais.

On peut estimer aujourd'hui qu'un système crypto nécessitant un effort de 2^{80} peut être qualifié de système sûr[26].

5.2. De l'utilité des autres attaques

L'attaque exhaustive est certainement l'attaque la plus simple à réaliser.

Il suffit d'obtenir un couple (texte clair, texte chiffré) ; de chiffrer le texte en clair avec toutes les clés possibles et de vérifier si le texte chiffré y correspond. Dès que l'opération réussit, l'attaque peut s'arrêter, la clé est trouvée.

Une autre variante de cette attaque consiste à prendre un texte en clair de le chiffrer avec toutes les clés possibles et de les stocker. Dès qu'une attaque doit être réalisée, il suffit de prendre le texte en clair, recueillir le résultat chiffré et retrouver dans la masse stockée[27] à quelle clé ce texte correspond.

Remarquons qu'il s'agit ici également d'un compromis mémoire/calcul.

De cette simplicité, il ressort que les attaques plus complexes comme l'analyse linéaire, l'analyse différentielle ou la « square attack » que nous allons évoquer plus loin n'ont d'utilité que si elle procure un avantage **substantiel** sur l'attaque exhaustive.

---

[26] Interview avec Monsieur Vincent Rijmen du 05 mai 2001

[27] Dénommée « the entire codebook »dans la littérature.

Une des pratiques utilisées est donc d'essayer ces attaques plus complexes sur un nombre réduit de round et d'augmenter progressivement le nombre de rounds et d'arrêter l'attaque dès que celle-ci atteint le niveau de complexité de l'attaque exhaustive.

Le nombre de rounds auxquels l'attaque est arrivée donne une indication toute relative mais réelle de la marge de sécurité en nombre de rounds.

5.3. La résistance à l'attaque différentielle

L'attaque différentielle d'une version limitée à 4 rounds nécessite 2^{150} mots choisis, ce qui est déjà de loin supérieur aux efforts nécessaires à l'attaque exhaustive.

Il faut dire que l'algorithme a été spécialement bien étudié pour résister à ce genre d'attaque. Parcourons les étapes de l'algorithme et les sécurités de chaque composante de l'algorithme[28].

5.3.1. La conception des S-Box

La S-Box est la seule composante non linéaire de l'algorithme (c'est elle qui apporte la confusion).

Remarquons que la découverte de l'inverse associée à la transformation affine n'est pas triviale. De description simple mais d'expression algébrique compliquée, elle peut-être vue comme une multiplication polynomiale suivie d'une addition.

$(x^7 + x^6 + x^2 + x) + a(x)\ (x^7 + x^6 + x^5 + x^4 + 1)$ modulo $x^8 + 1$ [29]

La transformation affine enlève la symétrie.

Les tests effectués dans la conception des S-Box sont l'absence de Fixed Point, c'est-à-dire qu'aucun byte ne peut être substitué à lui-même, et l'absence d'Opposite Fixed Point, c'est-à-dire qu'aucun byte n'est substitué à son complément.

Remarquons que dans le DES, les critères de conception des S-Box sont pour le moins nébuleux.

5.3.2. La diffusion en colonnes (MixColumn)

Les déterminants des matrices utilisées ne peuvent pas être nuls car cela constitue une faiblesse à la cryptanalyse différentielle.

Dans les transformations linéaires telle que la mixing column, une des mesures intéressantes de sécurité est le « branch number »:

---

[28] Ces caractéristiques servent également pour la résistance aux autres attaques.

[29] Selon l'AES proposal : Joan Daemen et Vincent Rijmen, *The Rijndael Block Cipher, KULeuven*, 1999, http://www.esat.kuleuven.ac.be/~rijmen/rijndael/, p26

Une différence dans 1 byte en input se propage sur les 4 bytes (c'est le principe même de la diffusion de l'information). Ce principe a été généralisé dans le RIJNDAEL.

Lors de la conception, les auteurs du RIJNDAEL ont tenu compte de cette sécurité de telle manière que pour des différences en input de x byte (de 1 à 4) la différence en output y soit maximale (soit telle que $x + y \supseteq 5$)

5.3.3. Key Schedule

L'utilisation des S-Boxes dans le « Key Schedule » augmente la sécurité de la clé ; en effet la découverte d'une clé de round ne nous donne certainement pas toute la clé originale (malheureusement le Key Schedule du RIJNDAEL diminue l'efficacité du système[30]).

En effet, une conception linéaire du key schedule aurait permis une plus grande efficacité dans le déchiffrement. Dans le cas du RIJNDAEL, toutes les clés doivent en effet être calculées avant de commencer à déchiffrer le premier byte (ce qui n'est pas le cas pour le DES).

La découverte de la clé d'un round du DES révèle 48 bits des 56 de la clé, l'algorithme est alors cassé, une recherche exhaustive sur les 8 bits restants est aisée et révèle la clé.

Toutefois, le Key Schedule du RIJNDAEL n'a pas toujours été jugé suffisant. En effet, lors des analyses pour le choix du nouveau standard, une classification des Key Schedule a été établie[31] . Cette classification en catégories et classes était la suivante :

- La **catégorie 1** désigne les algorithmes pour lesquels la connaissance d'une clé d'un round révèle une information sur les clés des rounds suivants ou sur la clé originale ('master Key').
- La **catégorie 2** désigne tout naturellement les algorithmes qui n'en révèlent pas.

Dans chaque catégorie des classes ont été définies.

- La **catégorie 1 type A** (1A) est utilisée lorsque tous les bits de la clé originale sont utilisés dans chaque round et donc lorsque la connaissance de la clé d'un round révèle *tous les bits* de la clé originale.
- La catégorie 1 **type B** s'emploie lorsque la connaissance de la clé d'un round donne *une partie des bits* de la clé originale.
- La catégorie 1 **type C** lorsque des fonctions d'inversions ou des *calculs arithmétiques simples sont nécessaires.*

---

[30] Dans une ancienne version du RIJNDAEL, le « Key Schedule » était totalement linéaire, les critiques sur ce point ont poussé les concepteurs à introduire une composante non linéaire à la partie clé de l'algorithme.

[31] G. Carter, E. Dawson, L. Nielsen, *Key Schedule Classification of the AES Candidates*, Queensland University of Technology, Queensland, Australia

- La **catégorie 2 type A** (2A) est utilisée *lorsqu'une partie des bits* de la clé originale sont utilisés dans chaque round
- La catégorie 2 **type B** (2B) s'emploie pour classer les algorithmes qui utilisent *tous les bits de la clé* pour construire la clé d'un round. L'entropie de la clé est alors maximisée.
- Les algorithmes les plus sûrs en ce qui concerne le Key Schedule sont ceux qui engendrent des clés indépendantes[32], ils sont classés en catégorie 2 **type C** (2C)

Le RIJNDAEL fut classé en catégorie 1C tandis que les principaux candidats (dont les quatre autres finalistes MARS – SERPENT – RC6 – TWOFISH) se trouvaient en catégories 2B. Le DES peut être classé en catégorie 1B.

5.4. La Square attack[33]

5.4.1. Principes de l'attaque

Cette attaque est plus longue que l'attaque différentielle. Il s'agit également d'une attaque à mots choisis qui s'effectue sur une version réduite du RIJNDAEL, en l'occurrence, un RIJNDAEL réduit à 4 round. On ne choisit pas ici des paires de textes en clair dont la différence est connue mais l'on considère 1 bloc de 1 byte parmi les 16 bytes.

Choisissons dans ce bloc 256 entrées où toutes les possibilités du premier byte sont représentées, il s'agit du « delta set », les autres bytes sont mis à 0, ce sont les « passive bytes ».

Fig 5.1 : Square Attack 1

Δset			

Examinons ensuite ce qui se passe étape par étape.

- Round initial

  Après l'**addition de la clé** du round initial, toutes les entrées possibles sont également représentées pour le premier byte, les autres bytes sont égaux et le XOR de ces 256 entrées est égal à 0.

[32] Cela veut dire que la longueur de la clé originale est égale à la somme des longueurs des clés de chaque round.

[33] Eli Biham et Nathan Keller, *Cryptanalysis of reduced variants of Rijndael*, 2000. http://csrc.ncsl.nist.gov/encryption/aes/round2/conf3/aes3papers.html

- Round 1

Après la première **substitution** (ByteSub), cette situation est toujours valable car la substitution remplace la valeur de chaque byte par une autre. Toutes les valeurs sont donc à nouveau représentées et le XOR des 256 textes est 0.

Le premier **décalage des lignes** est sans effet car la première ligne ne subit pas de décalage et les autres lignes représentent la substitution de 0 additionnée à la clé. Le XOR de ces valeurs est donc 0.

Après la première **diffusion en colonnes** (MixColumn) les sorties diffèrent dans les 4 bytes de la colonne où toutes les valeurs sont également représentées. En effet, prenons deux entrées différentes d'un byte du MixColumn, leur différence sera :

$$(02.\ (b1 - b1'),\ 01\ .\ (b2 - b2'),\ 01.\ (b3 - b3'),\ 03.\ (b4 - b4'))$$

La sortie différera donc dans les 4 bytes dans lesquels toutes les valeurs des bytes seront représentées.

Fig 5.2 : Square Attack 2

Δset			

L'**addition de la clé** du round 1 aura le même effet que la première addition de clé : les XOR de toutes les valeurs restera 0.

- Round 2

La même remarque vaut pour la **substitution** (ByteSub).

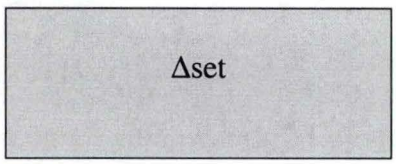
Le **décalage des lignes** (ShiftRow) du round 2 propagera les valeurs de la première colonne respectivement en position 1,2,3 et 4.

Fig 5.3 : Square Attack 3

Δset			
	Δset		
		Δset	
			Δset

La **diffusion en colonnes** (MixColumn) propagera le « delta set » dans tous les bytes du texte. Remarquons encore que l'opération XOR sur les 256 entrées donnera de nouveau 0.

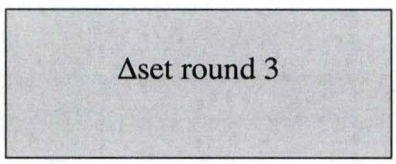
Fig 5.4 : Square Attack 4



- Round 3

  Puisqu'il s'agit de transformation linéaire et d'une substitution parmi les 256 valeurs possibles, la propriété du XOR restera vraie une dernière fois à la fin du round 3.

Fig 5.5 : Square Attack 5

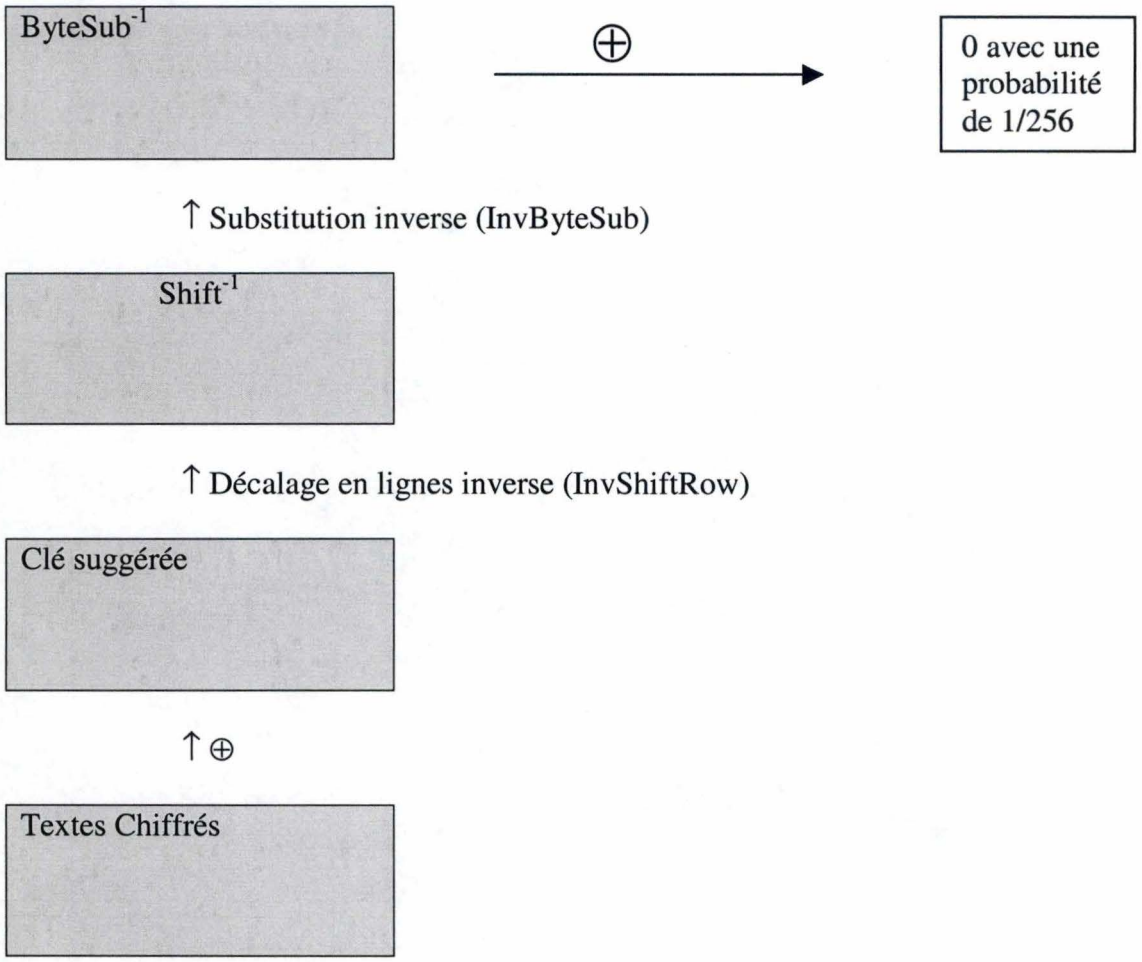


- Exécution de l'attaque

  Rappelons nous qu'il s'agit d'une variante du RIJNDAEL réduit à 4 rounds. Le round 4 dans cette variante est constitué de la substitution et de l'addition de la clé. L'attaque consiste à découvrir un byte de la clé du $4^{ème}$ round.

  Prenons maintenant les 256 textes chiffrés dans cette variante du RIJNDAEL et déchiffrons le dernier round pour le byte correspondant à la suggestion, nous obtiendrons les 256 sorties estimées du $3^{ème}$ round, s'il s'agit de la bonne suggestion, le XOR de ces 256 textes est 0.

Fig 5.6 : Square Attack 6



5.4.2. Extension de l'attaque et découverte de la clé originale

Des extensions de l'attaque (que nous ne détaillerons pas) à 5 et 6 voire 7 rounds ont été envisagées en ajoutant un round avant et un round après et en suggérant une partie de la clé. Mais aucune attaque supérieure à 7 rounds ne donnent une résultat plus intéressant que la recherche exhaustive.

Lorsque la clé d'un round a été découverte, des itérations supplémentaires pour retrouver la clé originale doivent encore être effectuées, c'est le « Reversed Key Schedule ». Le paragraphe ci-après résume l'effort nécessaire sur des variantes du RIJNDAEL à 4,5 ou 6 rounds, ce sont les meilleures attaques connues à l'heure actuelle sur le RIJNDAEL.

5.4.3. Mesure de la complexité de l'attaque

Attaque	Nb round	Nb de mots choisis	Complexité en temps
Square	4	2^9	2^9
Square	5	2^{11}	2^{40}
Square	6	2^{32}	2^{71}
Reversed key schedule	5	2^{11}	2^{31}
Reversed key schedule	6	2^{32}	2^{63}

5.5. L'attaque linéaire

Cette attaque ne sera pas étudiée en détail. Disons simplement qu'il s'agit de trouver une approximation linéaire qui décrit les transformations de l'algorithme.

Remarquons également que le DES est sensible à ce genre d'attaque. L'attaque linéaire sur le DES nécessite 2^{47} mots connus. Comparés au 2^{47} mots choisis de l'attaque différentielle, il s'agit d'une amélioration mineure puisqu'il est plus aisé d'acquérir des mots connus que des mots choisis.
Une attaque linéaire sur le RIJNDAEL limité à 8 rounds nécessitera au minimum 2^{150} mots connus.

5.6. L'existence de clés faibles

Comme nous l'avons vu précédemment, il existe dans le DES des clés faibles qui chiffrent le texte clair en un même texte chiffré. L'existence de clés faibles est une faiblesse. L'existence de ces clés nécessitent éventuellement[34] des opérations supplémentaires pour empêcher leur utilisation.

De par sa structure et son 'Key Scheduling', il n'est pas possible d'affirmer qu'il n'existe pas de clés faibles dans le RIJNDAEL mais tout porte d'ailleurs à le croire.
Admettons que, a priori, il n'existe pas de clé faible. (aucune clé faible n'a été trouvée jusqu'à présent.)

[34] car les probabilités sont faibles de générer ce genre de clé.

6. LE CRITÈRE D'EFFICACITE

Il s'agit de mesurer la vitesse de l'algorithme et la taille de la mémoire nécessaire sur des plates-formes différentes.

L'objectif de ce chapitre est de montrer comment le design de l'algorithme peut permettre une implémentation efficace en hardware, en software, flexible et simple. Nous allons passer en revue les différentes opérations des algorithmes DES et RIJNDAEL et les comparer. Les paramètres de comparaison sont les suivants :

- taille de la mémoire nécessaire
- nombre d'opérations simples
- Vitesse sur processeur 32-bits
- Vitesse sur processeur 64-bits
- Vitesse et faisabilité sur processeur 8-bits

6.1. Implémentation sur les processeurs 32-bits et 64-bits

Dans le RIJNDAEL, l'utilisation de mots de 32 bits permet d'augmenter l'efficacité de l'implémentation. Les étapes de la transformation par round peuvent être combinées.

De la même manière la diffusion en colonnes peut être traduite par une recherche dans une table de 16 mots de 8 bits en remplaçant la multiplication matricielle par une multiplication par les 4 vecteurs (de 32 bits) qui la composent.

Lors des tests des soumissionnaires à l'AES, les performances ont été comparées[35]. Les comparaisons portaient sur le temps de mise à clé et le temps nécessaires au chiffrement. Le nombre de blocs à chiffrer est également important et faisait aussi l'objet de comparaison.

Voici quelques chiffres de comparaison établis lors de l'analyse AES et rapporté dans les conclusions finales[36].

---

[35] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson, *Performance comparison of the AES Submissions*, February 1999.

[36] James Nechvatal, Elaine Barker, Lawrence Bassham, William Burr, Morris Dworkin, James Foti, Edward Roback, *Report ont the development of the AES*, NIST, 02 octobre 2000

Tableau 6.1 : Comparaison des vitesse de chiffrement des finalistes (en Kbits/sec) avec une clé de 128 Bits

	Processeur 32-Bits Intel Pentium Pro 200MHz en Java		
Algorithme	Chiffrement	Déchiffrement	Mise à clé
RIJNDAEL	19,621	18,868	96,234
SERPENT	11,464	11,519	34,729
MARS	19,718	19,443	28,680
TWOFISH	19,265	18,841	13,469
RC6	26,212	24,338	45,603

6.2. Conception pour Smart Card

Lors de l'appel à candidats, le NIST demanda une estimation des performances sur une plate-forme en 8 bits. Les réponses furent très variées, certains expliquèrent de manière détaillée comment implémenter l'algorithme sur Smart Card, d'autres de manière très vague[37]. MARS est, par exemple, presque impossible à implémenter sur Smart Card.

Ce critère a d'ailleurs pris une importance grandissante au cours des différentes phases de l'analyse par la NIST.

Les limitations à une telle application sont les suivantes :

- L'utilisation d'un code le plus restreint possible
- L'usage également le plus restreint des RAM (le moins de tables « look-up » possible)

La conception de l'algorithme RIJNDAEL traitant toutes les opérations par byte dans le champ Gallois fini GF(2^8) en font le meilleur candidat. Voici un tableau reprenant le vitesse de quelques algorithmes candidats à l'AES sur 2 Smart Card.

Tableau 6.2 : Vitesse de chiffrement sur Smart Card[38]

Algorithme	Smart 8051@ 3.57MHz	Smart ARM @ 28.56 MHz
E2	267 bytes/sec	44.142 bytes/sec
RC6	165 bytes/sec	151.260 bytes/sec
RIJNDAEL	3005 bytes/sec	311.492 bytes/sec
TWOFISH	Not Avalaible	56.289 bytes/sec

[37] Il faut dire que l'usage des smart-Card (d'invention française) est typiquement européen, proton, TV payantes... !

[38] Extrait de : Gaël Hachez, Frabçois Koeune, Jean-Jacques Quisquater, Caesar results : *Implementation of four AES Candidates on Two Smart Cards*, UCL, 4 février 1999, [www.dice.ucl.ac.be/crypto]

Les concepteurs du système RIJNDAEL ont également **privilégiés le chiffrement** en choisissant des matrices simples dans le chiffrement (multiplication par 1, 2 ou 3) même si l'inverse donne des coefficients plus élevés.
En effet, le chiffrement est plus utilisé que le déchiffrement : certaines opérations (notamment en smart card) ne nécessitent que le chiffrement (calcul MAC...).

6.3. Implémentation sur un processeur 8-bit

Parcourons les étapes du RIJNDAEL et étudions en l'implémentation :

- L'addition de la clé initiale :

Il s'agit de programmer un simple XOR effectué par byte, cette opération est élémentaire et peut être facilement implémentée en un nombre réduit de cycles

- Pour la substitution, elle s'effectue par byte (Champ Fini 2^8) et peut être programmée sans mémoire additionnelle ou implémentée par une 'table look-up'

- Il en va de même pour le décalage des lignes

- La diffusion en colonnes ; il s'agit d'une multiplication matricielle ; pour éviter l'attaque prenant en compte le temps de chiffrement (voir plus loin : timing attack), la multiplication sera implémentée par une table.

- L'expansion de la clé sera effectuée lorsque la clé est nécessaire (à chaque round). La sauvegarde de la clé dans une table n'est pas nécessaire si l'on ne veut chiffrer que quelques blocs. Il faudra donc conserver au minimum un buffer de 128, 192 ou 256 bits.

Le chiffrement a été implémenté sur les processeurs intel 8051 et Motorola 68HC08[39].

Rechercher l'inverse multiplicatif dans le champ (GF(256)) et une multiplication par byte. Ces 2 étapes peuvent être regroupées en une seule par une recherche dans une table de 256 mots de 8 bits.

[39] Selon l'AES proposal : Joan Daemen et Vincent Rijmen, *The Rijndael Block Cipher, KULeuven*, 1999, http://www.esat.kuleuven.ac.be/~rijmen/rijndael/ p23-24, les chiffres seront repris dans la partie de comparaison.

7. LES AUTRES CRITÈRES

Dans ce chapitre nous allons parcourir d'autres critères très importants étudiés lors de l'évaluation du RIJNDAEL.

7.1. Critères d'implémentation

Les critères d'implémentation ne sont pas au sens strict des critères qui s'appliquent sur l'algorithme lui-même mais sur son implémentation.

Il ne faut malgré tout pas s'étonner que ceux-ci soient pris en considération lors de l'évaluation d'un algorithme car le design de l'algorithme influence fortement la sensibilité de celui-ci à des attaques d'implémentation et les précautions qu'il faut prendre dépendent de cette conception.

7.1.1. La « Timing Attack »

7.1.1.1. Principe de l'attaque

Le principe de l'attaque est simple. Certaines opérations de l'algorithme peuvent prendre un temps différents (nombre de cycles machine) selon la valeur des opérandes.

L'attaque consiste à mesurer le temps de chiffrement et d'en déduire le contenu de l'information (la clé en l'occurrence).

7.1.1.2. Application au RIJNDAEL

La multiplication, telle que définie dans l'algorithme RIJNDAEL et utilisée dans la diffusion en colonnes, est la multiplication dans le champ fini « $GF(2^8)$ » correspondant à une multiplication polynomiale modulo un polynôme irréductible de degré 8.

Ce polynôme appelé m(x), choisi par les auteurs, est le polynôme binaire suivant :

$m(x) = x^8 + x^4 + x^3 + x^1 + 1$ soit `1 0001 1011` `'11B'`

Il en découle qu'un raccourci possible en implémentation (utilisant des opérations simples) pour la multiplication est le décalage vers la gauche suivi d'une addition avec 1B si le coefficient en x^8 est de 1. En effet la multiplication par x correspond à augmenter toutes les puissances de 1 et lorsque la puissance dépasse x^7, le polynôme du modulo intervient.

Prenons un exemple

Multiplions '42' par '75' et obtenons '42'.'75' = 'AE' :

$(x^6 + x^5 + x^4 + x^2 + 1) \bullet (x^6 + x) \rightarrow$

$(x^6 + x^5 + x^4 + x^2 + 1) \bullet x^6 = x^{12} + x^{11} + x^{10} + x^8 + x^6 +$

$x^7 + x^6 + x^5 + x^3 + x$

$= x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^3 + x$

$(x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^3 + x)$ modulo $(x^8 + x^4 + x^3 + x + 1)$

$x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^3 + x$

$(x^8 + x^4 + x^3 + x + 1) \bullet x^4 = x^{12} + x^8 + x^7 + x^5 + x^4$

$x^{11} + x^{10} + x^4 + x^3 + x$

$(x^8 + x^4 + x^3 + x + 1) \bullet x^3 = x^{11} + x^7 + x^6 + x^4 + x^3$

$x^{10} + x^7 + x^6 + x$

$(x^8 + x^4 + x^3 + x + 1) \bullet x^2 = x^{10} + x^6 + x^5 + x^3 + x^2$

$x^7 + x^5 + x^3 + x^2 + x$

ce qui donne (1010 1110) ou 'AE'

ou en implémentant le raccourci décrit plus haut :

```
'42'. '01' = '42'          0100 0010

'42'. '02' = '84'          1000 0100

'42'. '04' = '13'          0000 1000
                      ⊕    0001 1011
                           0001 0011

'42'. '08' = '26'          0010 0110

'42'. '10' = '4C'          0100 1100

'42'. '20' = '98'          1001 1000

'42'. '40' = '2B'          0011 0000
                      ⊕    0001 1011
                           0010 1011

'42'. '75' = '42' .('01' ⊕ '04' ⊕ '10' ⊕ '20' ⊕ '40') =
'AE'

Soit       0100 0010
     ⊕     0001 0011
```

```
⊕     0100 1100
⊕     1001 1000
⊕     0010 1011
=     1010 1110
```

Nous remarquons donc qu'en utilisant sans autre précaution cette manière d'implémenter la multiplication dans l'étape de diffusion en colonnes, il faut procéder à une addition supplémentaire (XOR) par '1B' si l'on découvre un report (coefficient du x^8).Cette opération aura donc une durée différente selon l'information à traiter tandis que les autres opérations (décalage, substitution et addition de la clé) dureront en moyenne un nombre de temps égal.

Cette attaque fut évoquée par les auteurs du RIJNDAEL[40] (prise en considération dans les implémentations décrites pour l'AES) et décrite par les professeurs François Koeune et Jean-Jacques Quisquater dans un rapport technique de 1999[41].

7.1.1.3. Conclusion

La conclusion n'est pas, comme l'ont indiqué les auteurs précités, une découverte d'une faiblesse du RIJNDAEL mais bien un avertissement indiquant avec quelles précautions une implémentation doit être envisagée sous peine d'affaiblir une sécurité à priori très forte.

Leurs expériences ont montré qu'avec « 3000 échantillons par byte de clé, la clé fut entièrement découverte à un coût négligeable sans optimiser l'attaque en question ».

Dans ce cas il faut donc veiller à ce que les opérations implémentées prennent un nombre constant de cycles indépendamment de la valeur de l'information à chiffrer ou de la valeur de la clé.

Cela pourra s'implémenter dans le RIJNDAEL en remplaçant l'opération décrite plus haut par une table ou en ajoutant à chaque fois soit '1B' soit '00' selon le cas pour s'assurer que le nombre de cycles nécessaires restent constants.

---

[40] AES proposal page 17

[41] François Koeune et Jean-Jacques Quisquater, *A timing attack against Rijndael, Technical Report*, CG-1999/1, UCL, http://www.dice.ucl.ac.be/crypto

7.1.2. La « Power Attack »

7.1.2.1. Principes de l'attaque

L'analyse s'effectue ici sur la consommation de courant. Si aucune précaution n'est prise, des instructions différentes consommeront des puissances différentes :

- l'addition
- Les opérations booléennes
- Charger une valeur dans un registre en RAM
- ...

L'adversaire pourra découvrir des informations sur la clé en mesurant la consommation, si la séquences des informations est dépendantes de la clé ou si la consommation est dépendante de la valeur des opérandes.

Par exemple si le DES n'est pas implémenté avec précaution, lors du Key Schedule, une analyse sur l'opération PC-2 (Permutated Choice 2) permet de retrouver tous les bits de la clé.

Il faut donc que la séquence des instructions soit indépendante de la clé[42] et que la consommation soit indépendante de la valeur des opérandes.

7.1.2.2. Application aux candidats AES

Hormis les XOR, voici un tableau reprenant les 5 candidats finalistes AES et les opérations effectuées par ceux-ci.

Tableau 7.1 : Opérations utilisées par les algorithmes[43]

Opération	RIJNDAEL	MARS	SERPENT	TWOFISH	RC6
Table / lookup	8 à 32	8 à 32		8 à 32	
Shift / Rotate	32	32	32	32	32
Logical operation		And Or	And Or Not...		And Fix
Addition		32		32	32
Substraction		32			32
Mutiplication		32			
Other					squaring mod2^{32}

[42] pour des instructions de consommations différentes.

[43] Extrait de : Joan Daemen et Vincent Rijmen, *Resistance Against Implementation Attack A comparative Study of the AES Proposals*, K.U.Leuven, 1 février 1999

7.1.2.3. Conclusion

Plus il existe d'opérations de consommations différentes, plus l'algorithme est difficile à protéger contre cette attaque sur l'implémentation. Moins les instructions nécessaires au chiffrement consomment, plus l'analyse est compliquée

---

La protection consiste, par exemple, à ajouter du bruit dans la consommation de puissance ou d'autres mesures telles que la désynchronisation ou le « Software Balancing » que nous n'allons pas détailler[44] .Les algorithmes les plus difficiles à protéger sont MARS et RC6, ensuite vient TWOFISH. Les plus favorables à sécuriser sont le RIJNDAEL et RC6.

7.2. La clarté et l'efficacité des spécifications[45]

La clarté des spécifications n'est certainement pas un critère à négliger, les ambiguïtés à ce niveau peuvent conduire à des incompréhensions et à des implémentations erronées.

Les spécifications ont été évaluées sur leur clarté sémantique. Les spécifications peuvent utiliser un style mathématique, du texte, des diagrammes ou encore du pseudo code.. Il est préférable d'utiliser une combinaison de styles et éviter trop de redondances qui augmentent le risque d'incohérence.

A ce niveau, le RIJNDAEL fut considéré comme bien spécifié bien qu'utilisant des conventions différentes dans les spécifications. Une bonne guidance dans l'implémentation était également fournie.

7.3. La clarté des fondements mathématiques

De la même manière que le point précédent, l'algorithme RIJNDAEL est basé sur des structures et fondements mathématiques clairs.

D'autres algorithmes comme le MARS ou TWOFISH utilisaient des mathématiques très compliqués et très difficiles à comprendre même pour des spécialistes en la matière.

Ceci est en fait une faiblesse, elle contredit l'idée même de la publicité de l'algorithme et de l'évolutivité de l'algorithme (changement des S-Box, augmentation du nombre de round, version modifiée pour des besoins spécifiques...)

---

[44] Une étude complète : Paul Kocher, Josua Jaffe et Benjamin Jim, *Differential Power Analysis*, Cryptography research Inc, San Francisco, www.cryptography.com

[45] Dr B.R. Gladman, *Implementation experience with AES algorithm*, 28th Feb 99

8. CONCLUSIONS

Au terme de ce mémoire, évaluons dans quelle mesure les objectifs assignés ont été atteints. Tirons-en les conclusions à en tirer et terminons par des considérations plus générales sur cette étude.

Le premier de ces objectifs était de décrire les critères de choix d'un bon algorithme. Ces critères se regroupent en 3 grandes catégories : la solidité cryptographique, la performance et les autres critères.

Après une brève description du DES et une description détaillée du RIJNDAEL, nous avons exposé le critère de la solidité cryptographique. Ce critère s'évalue en termes de résistance aux attaques connues sur l'algorithme ou sur une version plus réduite de cet algorithme ainsi qu'en termes de comparaison avec l'attaque de base ou l'attaque exhaustive.
L'attaque exhaustive ou brutale est du type le plus simple. Elle a été évoquée dans le chapitre historique et quantifiée pour le DES et pour RIJNDAEL[46].

Nous avons ensuite décrit l'attaque différentielle dans le chapitre consacré au DES et dans celui consacré à la solidité cryptographique. Nous avons montré comment l'algorithme RIJNDAEL s'est protégé avec succès contre cette attaque qui n'est pas intéressante même sur une version du RIJNDAEL réduite à 4 rounds.

Nous nous sommes ensuite penchés sur une attaque nouvelle inspirée de l'attaque différentielle, la « square attack ». Cette attaque s'effectue, en principe, sur une version également réduite à 4 rounds du RIJNDAEL. Elle peut être étendue à un plus grand nombre de rounds, mais retenons qu'elle n'est plus intéressante au 7ème round, ce qui donne au RIJNDAEL une marge de sécurité très satisfaisante.

Nous pouvons considérer que, pour ce qui concerne l'analyse de la solidité cryptographique, l'objectif a été en grande partie atteint ; en effet les principales attaques qui ont d'ailleurs été étudiées lors du processus d'évaluation sont reprises dans ce mémoire. Toutefois des limitations ont été apportées à l'étude ; elle sont essentiellement mathématiques. L'attaque linéaire par exemple, d'une complexité mathématique qui nous a paru trop élevée, a simplement été mentionnée.
Il en va de même pour bien d'autres attaques[47], principalement des variantes et des optimisations des attaques linéaires et différentielles.

La performance à savoir le critère d'efficacité est le sujet du chapitre 6. Dans ce chapitre, quelques chiffres intéressants sur les vitesses des algorithmes ont été repris.
Il en résulte que le RIJNDAEL est rapide et que, par sa structure, il permet un chiffrement sur Smart Card aisé et efficient.

---

[46] Cette mesure vaut également sur tous les algorithmes candidats puisque les longueurs des clés ont été spécifiées dans le cahier des charges (128, 192 et 256 bits).
[47] D'autres d'attaques sur les candidats AES : NIST, *Fast Software Encryption Workshop 2000*, New York, 10-12 avril 2000.

Le chapitre 7 développe les autres critères d'évaluation.
Le RIJNDAEL est bien spécifié et ses fondements mathématiques sont clairs.

Deux types attaques sur l'implémentation d'un algorithme ont également été étudiés. L'étude de ces deux attaques est nécessaire lorsqu'il s'agit non seulement de choisir mais aussi d'implémenter un algorithme ; c'est le deuxième objectif fixé.

Nous avons ensuite montré pourquoi certains « raccourcis » d'implémentation peuvent nuire à la sécurité. Dans le cas de la « Timing Attack », il faut en effet veiller à ce que le temps de chiffrement (ou de déchiffrement) ne soit pas dépendant de la valeur de la clé ni de l'information à chiffrer. C'est le cas si l'on implémente l'opération de multiplication (•) du RIJNDAEL par un décalage vers la gauche suivi d'une addition avec '1B' si le coefficient en x^8 est de 1.

Nous avons finalement attiré l'attention du lecteur sur une fragilité d'implémentation assez inattendue, à savoir la consommation de puissance des composants électroniques. C'est la base de la « Power Attack ». Cette attaque (qui nécessite toutefois un accès physique) recouvre la clé en étudiant, dans les différents rounds, les opérations effectuées et la consommation de courant y associée.
En ce qui concerne le choix d'un algorithme, retenons que celui qui utilise l'éventail le moins élevé d'opérations de consommations différentes est plus résistant à ce genre d'attaque. Retenons également que plus la consommation est faible, plus les différents « patterns » de consommation sont difficiles à déceler.
Au sujet de l'implémentation, notons que du bruit et/ou des « opérations blanches » peuvent être ajoutés pour uniformiser la consommation de courant lors des opérations de chiffrement ou de déchiffrement.

De façon plus générale, il est intéressant de s'interroger sur l'avenir du RIJNDAEL. Dans quelle mesure remplacera-t-il le DES et à quel rythme ? Quelle pourrait être son espérance de vie ?

Les applications existantes qui utilisent déjà le DES, comme par exemple les cartes de paiement n'implémenteront pas le RIJNDAEL dans un premier temps, le coût en serait trop important.
Par contre dans les nouvelles applications, l'algorithme RIJNDAEL sera, sans aucun doute utilisé. Il sera, par exemple, utilisé pour la nouvelle génération de GSM.
Ceci vaut également pour les applications pour lesquelles un changement d'algorithme ne nécessite pas beaucoup d'investissements ou lorsque, vu la faible vitesse du DES, l'investissement sera rentabilisé par le gain en rapidité du RIJNDAEL.

La période d'espérance de vie du RIJNDAEL n'est pas connue. Cet algorithme public est sujet à révision quinquennale. Toutefois on peut raisonnablement tabler sur une espérance d'exploitation de 20 à 30 ans.

Cryptographiquement comme physiquement, la protection parfaite n'existe pas. L'escalade entre les moyens de défense et l'attaque de ces moyens est une constante historique et rien ne peut laisser supposer que cela changera. Le problème est de

pouvoir établir le meilleur équilibre possible entre les valeurs à protéger et le prix des moyens que l'on est prêt à supporter pour cette protection.

Dans la situation actuelle, c'est bien ce à quoi le RIJNDAEL semble être parvenu.

Souhaitons un succès durable à cette conception belge de Messieurs Vincent RIJMEN et Joan DAEMEN qui ont remporté haut la main le concours AES .

9. BIBLIOGRAPHIE

9.1. Bibliographie générale

E. Biham et A Shamir, *Differential cryptanalysis of the Data Encryption Standard*, Springer-Verlag, New York, 1993

Bruce Schneier, *Cryptographie appliquée*, International Thomson Publishing, Paris, 1995

G. J. Simmons, *Contemporary Cryptology, The Science of Information Integrity*, IEEE press, Piscataway, NJ, 1992

William Stallings, Cryptography and Network Security, Prentice Hall, Upper Saddle River, New Jersey, 1995

Douglas R. Stinson, *Cryptography Theory and Practice*, CRC Press, Boca Raton, 1995.

9.2. Sites consultés et papiers

Ross Anderson, Eli Biham and Lars Knudsen, The case for Serpent, 24 March 2000, http://csrc.nist.gov/encryption/aes/round2

E. Biham et A. Shamir, *Power Analysis of the Key Scheduling of the AES Candidates*, Computer Science Department, Institute of Science, Israel, 2000

G. Carter, E. Dawson, L. Nielsen, *Key Schedule Classification of the AES Candidates*, Queensland University of Technology, Queensland, Australia

Site officiel du NIST pour la selection de l'AES. CSRC : Computer Security Division Information Technology Laboratory of National Institute of Standards and Technology, http://csrc.nist.gov/encryption/aes/

Joan Daemen et Vincent Rijmen, Answer to « new observations on Rijndael », *comparative Study of the AES Proposals*, K.U.Leuven, http://www.esat.kuleuven.ac.be/~rijmen/rijndael/

Joan Daemen et Vincent Rijmen, *Resistance Against Implementation Attack A comparative Study of the AES Proposals*, K.U.Leuven, 1 février 1999 http://www.esat.kuleuven.ac.be/~rijmen/rijndael/

Joan Daemen et Vincent Rijmen, *The Rijndael Block Cipher*, KULeuven, 1999, http://www.esat.kuleuven.ac.be/~rijmen/rijndael/

Henri Gilbert et Marine Minier, *A collision attack on 7 rounds of Rijndael, France Télécom R&D*, Issy les Moulineaux, France, http://csrc.ncsl.nist.gov/encryption/aes/round2/conf3/aes3agenda.html

Dr. Brian Gladman, *A specification for Rijndael, the AES Algorithm*, 3 mars 2001 http://fp.gladman.plus.com/cryptography_technology/aes/

Dr. Brian Gladman, *Implementation experience with AES algorithm*, 28th Feb 99 http://fp.gladman.plus.com/cryptography_technology/aes/

Gaël Hachez, François Koeune, Jean-Jacques Quisquater, Caesar results, *Implementation of four AES Candidates on Two Smart Cards*, UCL, 4 février 1999, www.dice.ucl.ac.be/crypto

Paul Kocher *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS and Other Systems*, Cryptography research Inc, San Francisco, www.cryptography.com

Paul Kocher, Josua Jaffe et Benjamin Jim, *Differential Power Analysis*, Cryptography research Inc, San Francisco, www.cryptography.com

François Koeune et Jean-Jacques Quisquater, *A timing attack against Rijndael, Technical Report*, CG-1999/1, UCL, http://www.dice.ucl.ac.be/crypto

Sean Murphy and Matt Robshaw, New observations on Rijndael, Information Security Group, University of London, UK, 7 aug 2000

James Nechvatal, Elaine Barker, Lawrence Bassham, William Burr, Morris Dworkin, James Foti, Edward Roback, *Report ont the development of the AES*, NIST, 02 octobre 2000, http://csrc.nist.gov/encryption/aes/

Vincent Rijmen, *Efficient Implementation of the Rijndael S-Box*, Katholieke Universiteit Leuven, Dept ESAT, http://www.esat.kuleuven.ac.be/~rijmen/rijndael/

Ronald Rivest, MJB Robshaw and Yiqun Lisa Yin, *RC6 as the AES*, M.I.T. Laboratory for Computer Science, 2000

Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson, *Comment on Twofish as an AES candidates*, 24 March 2000, http://csrc.nist.gov/encryption/aes/round1/conf2/aes2conf.htm

Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson, *Performance comparison of the AES Submissions*, February 1999, http://csrc.nist.gov/encryption/aes/round1/conf2/aes2conf.htm

Claude Shannon, *Communication Theory of Secrecy System*, Bell Technical Journal, 1949

10. ANNEXES

Annexe A : Les données du DES

The initial permutation IP is as follows:

IP							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

This means that the 58th bit of x is the first bit of $\mathrm{IP}(x)$; the 50th bit of x is the second bit of $\mathrm{IP}(x)$, etc.

The inverse permutation IP^{-1} is:

IP^{-1}							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

The expansion function E is specified by the following table:

E bit-selection table					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

The eight S-boxes and the permutation P are now presented:

S_1															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3															
10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4															
7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_5															
2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6															
12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7															
4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_8															
13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

P			
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Finally, we need to describe the computation of the key schedule from the key K. Actually, K is a bitstring of length 64, of which 56 bits comprise the key and 8 bits are parity-check bits (for error-detection). The bits in positions $8, 16, \ldots, 64$ are defined so that each byte contains an odd number of 1's. Hence, a single error can be detected within each group of 8 bits. The parity-check bits are ignored in the computation of the key schedule.

1. Given a 64-bit key K, discard the parity-check bits and permute the remaining bits of K according to a (fixed) permutation PC-1. We will write PC-1$(K) = C_0 D_0$, where C_0 comprises the first 28 bits of PC-1(K) and D_0 the last 28 bits.
2. For i ranging from 1 to 16, compute

$$C_i = LS_i(C_{i-1})$$
$$D_i = LS_i(D_{i-1}),$$

and $K_i = \text{PC-2}(C_i D_i)$. LS_i represents a cyclic shift (to the left) of either one or two positions, depending on the value of i: shift one position if $i = 1, 2, 9$ or 16, and shift two positions otherwise. PC-2 is another fixed permutation.

The key schedule computation is depicted in Figure 3.3.

The permutations PC-1 and PC-2 used in the key schedule computation are as follows:

PC-1						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

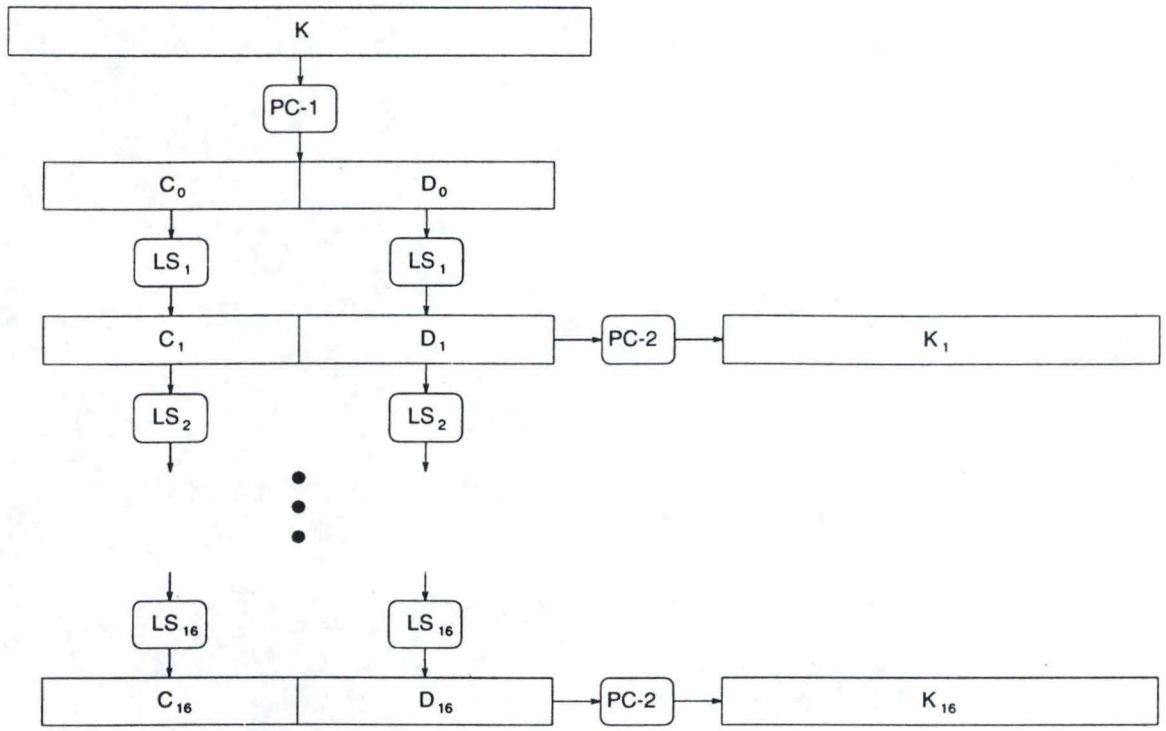


FIGURE 3.3
Computation of DES key schedule

PC-2					
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

We now display the resulting key schedule. As mentioned above, each round uses a 48-bit key comprised of 48 of the bits in K. The entries in the tables below refer to the bits in K that are used in the various rounds.

Round 1											
10	51	34	60	49	17	33	57	2	9	19	42
3	35	26	25	44	58	59	1	36	27	18	41
22	28	39	54	37	4	47	30	5	53	23	29
61	21	38	63	15	20	45	14	13	62	55	31

Round 2											
2	43	26	52	41	9	25	49	59	1	11	34
60	27	18	17	36	50	51	58	57	19	10	33
14	20	31	46	29	63	39	22	28	45	15	21
53	13	30	55	7	12	37	6	5	54	47	23

Round 3											
51	27	10	36	25	58	9	33	43	50	60	18
44	11	2	1	49	34	35	42	41	3	59	17
61	4	15	30	13	47	23	6	12	29	62	5
37	28	14	39	54	63	21	53	20	38	31	7

Round 4											
35	11	59	49	9	42	58	17	27	34	44	2
57	60	51	50	33	18	19	26	25	52	43	1
45	55	62	14	28	31	7	53	63	13	46	20
21	12	61	23	38	47	5	37	4	22	15	54

Round 5											
19	60	43	33	58	26	42	1	11	18	57	51
41	44	35	34	17	2	3	10	9	36	27	50
29	39	46	61	12	15	54	37	47	28	30	4
5	63	45	7	22	31	20	21	55	6	62	38

Round 6											
3	44	27	17	42	10	26	50	60	2	41	35
25	57	19	18	1	51	52	59	58	49	11	34
13	23	30	45	63	62	38	21	31	12	14	55
20	47	29	54	6	15	4	5	39	53	46	22

Round 7											
52	57	11	1	26	59	10	34	44	51	25	19
9	41	3	2	50	35	36	43	42	33	60	18
28	7	14	29	47	46	22	5	15	63	61	39
4	31	13	38	53	62	55	20	23	37	30	6

Round 8											
36	41	60	50	10	43	59	18	57	35	9	3
58	25	52	51	34	19	49	27	26	17	44	2
12	54	61	13	31	30	6	20	62	47	45	23
55	15	28	22	37	46	39	4	7	21	14	53

Round 9											
57	33	52	42	2	35	51	10	49	27	1	60
50	17	44	43	26	11	41	19	18	9	36	59
4	46	53	5	23	22	61	12	54	39	37	15
47	7	20	14	29	38	31	63	62	13	6	45

Round 10											
41	17	36	26	51	19	35	59	33	11	50	44
34	1	57	27	10	60	25	3	2	58	49	43
55	30	37	20	7	6	45	63	38	23	21	62
31	54	4	61	13	22	15	47	46	28	53	29

Round 11											
25	1	49	10	35	3	19	43	17	60	34	57
18	50	41	11	59	44	9	52	51	42	33	27
39	14	21	4	54	53	29	47	22	7	5	46
15	38	55	45	28	6	62	31	30	12	37	13

Round 12											
9	50	33	59	19	52	3	27	1	44	18	41
2	34	25	60	43	57	58	36	35	26	17	11
23	61	5	55	38	37	13	31	6	54	20	30
62	22	39	29	12	53	46	15	14	63	21	28

Round 13											
58	34	17	43	3	36	52	11	50	57	2	25
51	18	9	44	27	41	42	49	19	10	1	60
7	45	20	39	22	21	28	15	53	38	4	14
46	6	23	13	63	37	30	62	61	47	5	12

Round 14											
42	18	1	27	52	49	36	60	34	41	51	9
35	2	58	57	11	25	26	33	3	59	50	44
54	29	4	23	6	5	12	62	37	22	55	61
30	53	7	28	47	21	14	46	45	31	20	63

Round 15											
26	2	50	11	36	33	49	44	18	25	35	58
19	51	42	41	60	9	10	17	52	43	34	57
38	13	55	7	53	20	63	46	21	6	39	45
14	37	54	12	31	5	61	30	29	15	4	47

Round 16											
18	59	42	3	57	25	41	36	10	17	27	50
11	43	34	33	52	1	2	9	44	35	26	49
30	5	47	62	45	12	55	38	13	61	31	37
6	29	46	4	23	28	53	22	21	7	63	39

Decryption is done using the same algorithm as encryption, starting with y as the input, but using the key schedule $K_{16}, \ldots, K_1$ in reverse order. The output will be the plaintext x.

Annexe B : Cryptanalyse différentielle du DES réduit à 6 rounds.

FIGURE 3.13
Differential attack on 6-round DES

Input: $L_0R_0, L_0^*R_0^*, L_6R_6$ and $L_6^*R_6^*$, where $L_0' = 40080000_{16}$ and $R_0' = 04000000_{16}$

1. compute $C' = \mathrm{P}^{-1}(R_6' \oplus 04000000_{16})$
2. compute $E = \mathrm{E}(L_6)$ and $E^* = \mathrm{E}(L_6^*)$
3. **for** $j \in \{2, 5, 6, 7, 8\}$ **do**
   compute $test_j(E_j, E_j^*, C_j')$.

We expect that about $1/16$ of our pairs are right pairs and the rest are wrong pairs with respect to our 3-round characteristic.

Our strategy is to compute E_j, E_j^*, and C_j', as described above, and then to determine $test_j(E_j, E_j^*, C_j')$, for $j = 2, 5, 6, 7, 8$. If we start with a right pair, then the correct key bits for each J_j will be included in the set $test_j$. If the pair is a wrong pair, then the value of C_j' will be incorrect, and it seems reasonable to hypothesize that each set $test_j$ will be essentially random.

We can often identify a wrong pair by this method: If $|test_j| = 0$, for any $j \in \{2, 5, 6, 7, 8\}$, then we necessarily have a wrong pair. Now, given a wrong pair, we might expect that the probability that $|test_j| = 0$ for a particular j is approximately $1/5$. This is a reasonable assumption since $N_j(E_j', C_j') = |test_j|$ and, as mentioned earlier, the probability that $N_j(E_j', C_j') = 0$ is approximately $1/5$. The probability that all five $test_j$'s have positive cardinality is estimated to be $.8^5 \approx .33$, so the probability that at least one $test_j$ has zero cardinality is about .67. So we expect to eliminate about $2/3$ of the wrong pairs by this simple observation, which we call the *filtering operation*. The proportion of right pairs that remain after filtering is approximately

$$\frac{\frac{1}{16}}{\frac{1}{16} + \frac{15}{16} \times \frac{1}{3}} = \frac{1}{6}.$$

Example 3.4

Suppose we have the following plaintext-ciphertext pair:

plaintext	ciphertext
86FA1C2B1F51D3BE	1E23ED7F2F553971
C6F21C2B1B51D3BE	296DE2B687AC6340

Observe that $L'_0 = 40080000_{16}$ and $R'_0 = 04000000_{16}$. The S-box inputs and outputs for round 6 are computed to be the following:

j	E_j	E_j^*	C'_j
2	111100	010010	1101
5	111101	111100	0001
6	011010	000101	0010
7	101111	010110	1100
8	111110	101100	1101

Then, the sets $test_j$ are as follows:

j	$test_j$
2	14, 15, 26, 30, 32, 33, 48, 52
5	
6	7, 24, 36, 41, 54, 59
7	
8	34, 35, 48, 49

We see that both $test_5$ and $test_7$ are empty sets, so this pair is a wrong pair and is discarded by the filtering operation. □

Now suppose that we have a pair such that $|test_j| > 0$ for $j = 2, 5, 6, 7, 8$, so that it survives the filtering operation. (Of course, we do not know if the pair is a right pair or a wrong pair.) We say that the bitstring $J_2 J_5 J_6 J_7 J_8$ of length 30 is *suggested* by the pair if $J_j \in test_j$ for $j = 2, 5, 6, 7, 8$. The number of suggested bitstrings is

$$\prod_{j \in \{2,5,6,7,8\}} |test_j|.$$

It is not unusual for the number of suggested bitstrings to be quite large (for example, greater than 80000).

Suppose we were to tabulate all the suggested bitstrings obtained from the N pairs that were not discarded by the filtering operation. For every right pair, the correct bitstring $J_2 J_5 J_6 J_7 J_8$ will be a suggested bitstring. This correct bitstring will be counted about $3N/16$ times. Incorrect bitstrings should occur much less often, since they will occur essentially at random and there are 2^{30} possibilities (a very large number).

It would get extremely unwieldy to tabulate all the suggested bitstrings, so we use an algorithm that requires less space and time. We can encode any $test_j$ as a vector T_j of length 64, where the ith coordinate of T_j is set to 1 (for $0 \leq i \leq 63$) if the bitstring of length six that is the binary representation of i is in the set $test_j$; and the ith coordinate is set to 0 otherwise (this is essentially the same as the counter array representation that we used in the 3-round attack).

For each remaining pair, construct these vectors as described above, and name them T_j^i, $j = 2, 5, 6, 7, 8$, $1 \leq i \leq N$. For $I \subseteq \{1, \ldots, N\}$, we say that I is *allowable* if for each $j \in \{2, 5, 6, 7, 8\}$, there is at least one coordinate equal to $|I|$ in the vector

$$\sum_{i \in I} T_j^i.$$

If the ith pair is a right pair for every $i \in I$, then the set I is allowable. Hence, we expect there to be an allowable set of size (approximately) $3N/16$, which we hope will suggest the correct key bits and no other. It is a simple matter to construct all the allowable sets I by means of a recursive algorithm.

Example 3.5

We did some computer runs to test this approach. A random sample of 120 pairs of plaintexts with the specified x-ors was generated, and these were encrypted using the same (random) key. We present the 120 pairs of ciphertexts and corresponding plaintexts in hexadecimal form in Table 3.1.

When we compute the allowable sets, we obtain n_i allowable sets of cardinality i, for the following values:

i	n_i
2	111
3	180
4	231
5	255
6	210
7	120
8	45
9	10
10	1

The unique allowable set of size 10 is

$$\{24, 29, 30, 48, 50, 52, 55, 83, 92, 118\}.$$

In fact, it does arise from the 10 right pairs. This allowable set suggests the correct key bits for J_2, J_5, J_6, J_7 and J_8 and no others. They are as follows:

$$J_2 = 011001$$

$$J_5 = 110000$$

$$J_6 = 001001$$

$$J_7 = 101010$$

$$J_8 = 100011$$

FIGURE 3.14
Another 3-round characteristic

L'_0	=	00200008_{16}	R'_0	=	00000400_{16}	
L'_1	=	00000400_{16}	R'_1	=	00000000_{16}	$p = 1/4$
L'_2	=	00000000_{16}	R'_2	=	00000400_{16}	$p = 1$
L'_3	=	00000400_{16}	R'_3	=	00200008_{16}	$p = 1/4$

Note that all the allowable sets of cardinality at least 6, and all but three of the allowable sets of cardinality 5, arise from right pairs, since $\binom{10}{5} = 252$ and $\binom{10}{i} = n_i$ for $6 \leq i \leq 10$.

This method yields 30 of the 56 key bits. By means of a different 3-round characteristic, presented in Figure 3.14, it is possible to compute 12 further key bits, namely those in J_1 and J_4. Now only 14 key bits remain unknown. Since $2^{14} = 16384$ is quite small, an exhaustive search can be used to determine the remaining 14 key bits.

The entire key (in hexadecimal, including parity-check bits) is:

```
34E9F71A20756231.
```

As mentioned above, the 120 pairs are given in Table 3.1. In the second column, a * denotes that a pair is a right pair, while a ** denotes that the pair is an identifiable wrong pair and is discarded by the filtering operation. Of the 120 pairs, 73 are identified as being wrong pairs by the filtering process, so 47 pairs remain as "possible" right pairs. □

3.6.3 Other examples of Differential Cryptanalysis

Differential cryptanalysis techniques can be used to attack **DES** with more than six rounds. An 8-round **DES** requires 2^{14} chosen plaintexts, and 10-, 12-, 14- and 16-round **DES**s can be broken with 2^{24}, 2^{31}, 2^{39} and 2^{47} chosen plaintexts, respectively. The attacks on more than 10 rounds are probably not practical at this time.

Several substitution-permutation product ciphers other than **DES** are also susceptible (to varying degrees) to differential cryptanalysis. These cryptosystems include several substitution-premutation cryptosystems that have been proposed in recent years, such as FEAL, REDOC-II, and LOKI.

TABLE 3.1
Cryptanalysis of 6-round DES

pair	right pair?	plaintext	ciphertext
1	**	86FA1C2B1F51D3BE	1E23ED7F2F553971
		C6F21C2B1B51D3BE	296DE2B687AC6340
2	**	EDC439EC935E1ACD	0F847EFE90466588
		ADCC39EC975E1ACD	93E84839F374440B
3	**	9468A0BE00166155	3D6A906A6566D0BF
		D460A0BE04166155	3BC3B236398379E1
4	**	D4FF2B18A5A8AAC8	26B14738C2556BA4
		94F72B18A1A8AAC8	15753FDE86575A8F
5		09D0F2CF277AF54F	15751F4F11308114
		49D8F2CF237AF54F	6046A7C863F066AF
6		CBC7157240D415DF	7FCDC300FB9698E5
		8BCF157244D415DF	522185DD7E47D43A
7		0D4A1E84890981C1	E7C0B01E32557558
		4D421E848D0981C1	912C6341A69DF295
8	**	6CE6B2A9B8194835	75D52E028A5C48A3
		2CEEB2A9BC194835	6C88603B48E5A8CE
9	**	799F63C3C9322C1A	A6DA322B8F2444B5
		399763C3CD322C1A	6634AA9DF18307F4
10	**	1B36645E381EDF48	1F91E295D559091B
		5B3E645E3C1EDF48	D094FC12C02C17CA
11		85CA13F50B4ADBB9	ED108EE7397DDE0A
		C5C213F50F4ADBB9	3F405F4A3E254714
12	**	7963A8EFD15BC4A1	8C714399715A33BA
		396BA8EFD55BC4A1	C344C73CC97E4AC4
13		7BCFF7BCA455E65E	475A2D0459BCCE62
		3BC7F7BCA055E65E	8E94334AEF359EF8
14		0C505CEDB499218C	D3C66239E89CC076
		4C585CEDB099218C	9A316E801EE18EB1
15		6C5EA056CDC91A14	BC7EBA159BCA94E6
		2C56A056C9C91A14	67DB935C21FF1A8D
16	**	6622A441A0D32415	35F8616FEBA62883
		262AA441A4D32415	4313E1925F5B64BC
17		C0333C994AFF1C99	D46A4CF1C0221B11
		803B3C994EFF1C99	D22B42DB150E2CE8
18		9E7B2974F00E1A6E	172D286D9606E6FE
		DE732974F40E1A6E	2217A91F8C427D27
19	**	CF592897BFD70C7E	FB892B59E7DCE7EC
		8F512897BBD70C7E	C328B765E1CC6653
20		E976CF19124A9FA1	905BF24188509FA6
		A97ECF19164A9FA1	9ADDBA0C23DD724F
21	**	5C09696E7363675D	92D60E5C71801A99
		1C01696E7763675D	DD90908A4FE8168F
22	**	A8145AB3C1B2C7DE	F68FC9F80564847B
		E81C5AB3C5B2C7DE	51C041B5711B8132
23		47DF6A0BB1787159	52E36C4CA22EA5A2
		07D76A0BB5787159	373EAFD503F68DE4
24	*	7CE65464329B4E6D	832A9D7032015D9F
		3CEE5464369B4E6D	85E2CE665571E99C

pair	right pair?	plaintext	ciphertext
25	**	421FB6AD95791BA7	D1E730BA1DB565E7
		0217B6AD91791BA7	188E61735FA4F3CE
26	**	C58E9A361368FFD6	795EB9D30CAE6879
		85869A361768FFD6	26D37AC4867ACC61
27	**	DD86B6C74C8EA4E2	CC3B6915C9A348DF
		9D8EB6C7488EA4E2	104C2394555645F0
28	**	43DB9D2F483CA585	E3E4DA503D1B9396
		03D39D2F4C3CA585	4EA02C0061332443
29	*	855A309F96FEA5EA	85AD6E9E352AFAFA
		C552309F92FEA5EA	929D22370ACAB80D
30	*	AB3CA25B02BD18C8	0F7D768E9203F786
		EB34A25B06BD18C8	A1313BC26A99D353
31	**	A9F7A6F4A7C00E06	F26B385E6BA057FD
		E9FFA6F4A3C00E06	203D8384F8F54D19
32	**	688B9ACD856D1312	C41D99C107B4EF76
		28839ACD816D1312	6CC817CA025A7DAC
33	**	76BF0621C03D4CD9	BBE1F95AFC1E052A
		36B70621C43D4CD9	561F4801F2EB0C63
34	**	014CF8D1F981B8EE	D27091C4314CBFE8
		4144F8D1FD81B8EE	B7976D6A80E3DB61
35	**	487D66EDE0405F8C	8136325C0AEB84CE
		087566EDE4405F8C	8C638BC4495B69A0
36	**	DDCA47093A362521	51040CF16B600FAA
		9DC247093E362521	7FC75515AC3CAAF9
37	**	45A9D34A3996F6D9	F2004B854AE6C46C
		05A1D34A3D96F6D9	546825016B03D193
38	**	295D2FBFB00875EA	A309DF027E69C265
		69552FBFB40875EA	4F633FFB95A0C11E
39		964C8B98D590D524	1FF1D0271D6F6C18
		D6448B98D190D524	8CF2D8D401EBFC0F
40		60383D2BAF0836BC	10A82D55FC480640
		20303D2BAB0836BC	602346173581EF79
41	**	5CF8D539A22A1CAD	92685D806FBE8738
		1CF0D539A62A1CAD	17006DAB2D28081C
42		F95167CAB6565609	C52E2EB27446054E
		B95967CAB2565609	0C219F686840E57A
43		49F1C83615874122	2680C8ECDF5E51CD
		09F9C83611874122	5022A7B69B4E75EF
44	**	ACB2EC1941B03765	D6B593460098DEC5
		ECBAEC1945B03765	D3190A0200FC6B9B
45		CCCC129D5CB55EC0	3AD22B7EF59E0D5E
		8CC4129D58B55EC0	A48C92CBEC17E430
46	**	917FF8E2EE6B78D5	EF847E058DB71724
		D177F8E2EA6B78D5	F243F0554A00E4C5
47	**	51DBCF028E96DE00	574897CA1EE73885
		11D3CF028A96DE00	9F0FD0A5B2C2B5FD
48	*	2094942E093463CE	59F6A018C6A0D820
		609C942E0D3463CE	799FE001432346C0

pair	right pair?	plaintext	ciphertext
49	**	50FB0723D7CD1081	16AF758395EA3A7D
		10F30723D3CD1081	CDCB23392D144BED
50	*	740815A4F6CDCABB	4A84D2ED4D9351AB
		340015A4F2CDCABB	5923D04CE94D6111
51	**	EDA46A1AE93735DC	0B302A51B7E5476A
		ADAC6A1AED3735DC	5F817F0ABC770E75
52	*	08BC39B766B2C128	DFB5F3F500BC0100
		48B439B762B2C128	B7B9FED8AC93EBFA
53	**	A74E29BBA98F2312	A2B352B7F922E8DA
		E74629BBAD8F2312	D6BC4B89CED2DEAC
54	**	D6F50D31EE4E68AB	4D464847065C0938
		96FD0D31EA4E68AB	7554D87AEDCE5634
55	*	06191AA594891CF5	649C1D084F920F9E
		46111AA590891CF5	BE12A10384365E19
56		5EA7EFD557946962	15E664293F4D77EE
		1EAFEFD553946962	E23396A758DC9CE6
57	**	41FB7704781CC88A	8ABD385C441FD6CE
		01F377047C1CC88A	06DE8D55777AB65C
58	**	9689B9123F7C5431	E1E63120742099BB
		D681B9123B7C5431	1AF88A2CF6649A4A
59		6F25032B4A309BFE	48FE50DE774288D7
		2F2D032B4E309BFE	47950691260D5E10
60	**	D8C4B02D8E8BF1E9	F34D565E6AE85683
		98CCB02D8A8BF1E9	A4D2DB548622A8E8
61	**	F663E8CCEE86805B	51BD62C9D5D0F0BB
		B66BE8CCEA86805B	D2ABB03CF9D26C0A
62	**	428B29BFDFA838DB	006D62A65761089F
		028329BFDBA838DB	9FD73EF6124B0C11
63	**	04BE2D22D81EDC66	26D99536D99B5707
		44B62D22DC1EDC66	94144EBDA0CDEB55
64	**	667B779123A3EF80	5D09CBF2CE7E5A69
		2673779127A3EF80	5EFF8BFCA7BAA152
65	**	BC86D401D6572438	E05572AAA5F6C377
		FC8ED401D2572438	3C670BC455144F61
66	**	6FE5E9547659E401	2C465BF6F52F864C
		2FEDE9547259E401	B71D106444F95F31
67	**	27D3BAC6453BE3DE	8F160E29000461CD
		67DBBAC6413BE3DE	2A6660F46487F885
68	**	1D864E7642A7023A	65F91EEBFD8A9C05
		5D8E4E7646A7023A	84761791B3C36661
69	**	5256CA6894707CBA	91527F9349ABCF15
		125ECA6890707CBA	30F28F06A7B0A35A
70	**	C05383B8EFCD2BD7	710B6EC61BF63E9C
		805B83B8EBCD2BD7	53AC029D8E0179D5
71		50EB21CA13F9A96E	26D95BA4DE4C85CF
		10E321CA17F9A96E	8F01A90F638AFFF6
72	**	60EB1229ACD90EDC	3890EE8567782F96
		20E31229A8D90EDC	EE404DF7BE537589

pair	right pair?	plaintext	ciphertext
73		8E9A17D17B173B99	885C3933627EDEF0
		CE9217D17F173B99	B7ABB6DF5835E962
74		6EC5CD0802C98817	A985ADFB1FEE013C
		2ECDCD0806C98817	0428DE024B7E4604
75	**	1E81712FF1145C06	417E667A99B3CFA5
		5E89712FF5145C06	5C24AA056EB1ADBA
76	**	DF3C5C13311AEC7C	BF01675096F1C48A
		9F345C13351AEC7C	243D99BCE12DB864
77	**	7C34472994127C2D	713915DA311A7CF4
		3C3C472990127C2D	E9733D11D787E20B
78	**	37304DABA75EAFB3	EFB5C37FA0238ADF
		77384DABA35EAFB3	A728F7407AF958B3
79		D03A16E4C2D8B54B	423FC0AC24CEFEDD
		903216E4C6D8B54B	047D8595DB4D372E
80	**	8CED882B5D91832E	0006E2DE3AF5C2B5
		CCE5882B5991832E	00F6AA9ED614001B
81	**	1BB0E6C79EFBEC41	E9AED4363915775A
		5BB8E6C79AFBEC41	655BC48F1FFB5165
82		D41B8346DA9E2252	34F5E0BCC5B042EA
		94138346DE9E2252	702D2C48CDBE5173
83	*	02A9D0A0A91F6304	E2F1C10E59AF07C5
		42A1D0A0AD1F6304	BDEE6AA00F25F840
84	**	841B3E27C8F0A561	2B288E554D712C92
		C4133E27CCF0A561	FF8609C9E7301162
85	**	CDF0A8D6EE909185	5D661834D1C76324
		8DF8A8D6EA909185	22034D57D21FFB56
86	**	4C31AC854F44EA34	BD016309AEDB9BB1
		0C39AC854B44EA34	C72EEDC4FA1D9312
87		DB3FC0703C972930	296ABCFBF01DF991
		9B37C07038972930	CA4700686F9F83A2
88		E4B362BFD6A7CFD1	20FDAF335F25B1DA
		A4BB62BFD2A7CFD1	008C24D75E14ACBD
89		F234232A0E0A4A28	90CFD699F2DEC5BD
		B23C232A0A0A4A28	2918D3DE0C1B689C
90	**	71265345A5874004	3052CE3CE88710AE
		312E5345A1874004	38F0FC685DF30564
91	**	3E6364548C857110	0E8581E42C9FEC6F
		7E6B645488857110	4DD1751861EC5529
92	*	464FBEDBD78900A7	90F5F9ADEDED627A
		0647BEDBD38900A7	2EF4C540425E339B
93	**	373B75F847480BB0	5408B964F8442D16
		773375F843480BB0	805287D52599E9F0
94	**	D714E87810DE97AC	4EC4D623108FA909
		971CE87814DE97AC	0AA0725CED10D6A3
95		B9B5932EF54B2C60	4B438B3CCF36DEC9
		F9BD932EF14B2C60	054C6A337709280D
96	**	2F283C38D2E4E1DD	83515FB6DFEA90B8
		6F203C38D6E4E1DD	09BCC4FF38C78C23

pair	right pair?	plaintext	ciphertext
97	**	1EB8ADAA43BBD575	21A1E04813616E42
		5EB0ADAA47BBD575	D044BA3F25DFD02A
98	**	3164AA5454D9F991	9382C6C1883F1038
		716CAA5450D9F991	5CDFED4FF2117DEC
99		D78C1C5C6F2243D2	1CCEB091E030E6A6
		97841C5C6B2243D2	4DA2CD67CC449B21
100		BBE212A7D3CE3D14	2917C207B4D93E0D
		FBEA12A7D7CE3D14	A01D50E5A2B902D8
101	**	104917795E98D0FB	40916A71385C2803
		504117795A98D0FB	413FD26EF671F46D
102	**	4DDA114D6EFEEEB4	2E2C65E1D5CBAC31
		0DD2114D6AFEEEB4	A16FF03BC0913ED6
103		E0BED7B285BF0A77	5D9EFEFF0AD10490
		A0B6D7B281BF0A77	4C6CA1FAC36A8E5B
104	**	0AE1555FA1716214	378400BCED39EB81
		4AE9555FA5716214	A1E0C758BD8912C2
105	**	4657C26790FCB354	588BA079B2E7ED20
		065FC26794FCB354	DA90827AEED7A41F
106	**	32BD719B0DC1B091	F3477C7552BCB05D
		72B5719B09C1B091	EFF444449D66BE9E
107	**	0992F8C8C73A9BFE	9F3FFD0F158295F6
		499AF8C8C33A9BFE	C138358DCECC8FC7
108		02C3F061A237BBEB	AC28B0307127EA7C
		42CBF061A637BBEB	3FF1DAED9E0FCBC5
109	**	80E529E69EDE6827	1DF1DB7B66BA1AF1
		C0ED29E69ADE6827	15700151A5804549
110		B55E84630067B8D5	88321611FF9DA421
		F55684630467B8D5	90649D7EACF91F9A
111		2749C2EBC603BFF2	A62B23A7348E2C3A
		6741C2EBC203BFF2	EB760A09C7FF5153
112	**	C4C5E14D4C5D9FF5	ABC2312FBFD94DF5
		84CDE14D485D9FF5	D2BB5954E5062D53
113	**	1566BA21F2647E18	A247ED988457CB78
		556EBA21F6647E18	5E99F231005F5249
114	**	2D093D426D922F92	5DF62030B9F23AE9
		6D013D4269922F92	5D92DA1FA3D07BA1
115		004518468E0C96C3	F28D85FF7E84F38F
		404D18468A0C96C3	52541B0443053C57
116	**	437B70A98AE03344	04B3FBF9823B4CF7
		037370A98EE03344	14EBEC79DAD3093E
117		2D01F1073D3E375B	F10B3E1EE356226C
		6D09F107393E375B	6FF26DA5E3525B62
118	*	66573DD7E0D7F110	F2F26204C29FE51E
		265F3DD7E4D7F110	083A4ECE57E429AC
119		0846DB9538155201	F120D0D2AE788057
		484EDB953C155201	00CC914A33034782
120		ABB34FC195C820D1	5F17AE066B50FC81
		EBBB4FC191C820D1	2858DD63A2FA4B53

pair	right pair?	plaintext	ciphertext
97	**	1EB8ADAA43BBD575	21A1E04813616E42
		5EB0ADAA47BBD575	D044BA3F25DFD02A
98	**	3164AA5454D9F991	9382C6C1883F1038
		716CAA5450D9F991	5CDFED4FF2117DEC
99		D78C1C5C6F2243D2	1CCEB091E030E6A6
		97841C5C6B2243D2	4DA2CD67CC449B21
100		BBE212A7D3CE3D14	2917C207B4D93E0D
		FBEA12A7D7CE3D14	A01D50E5A2B902D8
101	**	104917795E98D0FB	40916A71385C2803
		504117795A98D0FB	413FD26EF671F46D
102	**	4DDA114D6EFEEEB4	2E2C65E1D5CBAC31
		0DD2114D6AFEEEB4	A16FF03BC0913ED6
103		E0BED7B285BF0A77	5D9EFEFF0AD10490
		A0B6D7B281BF0A77	4C6CA1FAC36A8E5B
104	**	0AE1555FA1716214	378400BCED39EB81
		4AE9555FA5716214	A1E0C758BD8912C2
105	**	4657C26790FCB354	588BA079B2E7ED20
		065FC26794FCB354	DA90827AEED7A41F
106	**	32BD719B0DC1B091	F3477C7552BCB05D
		72B5719B09C1B091	EFF444449D66BE9E
107	**	0992F8C8C73A9BFE	9F3FFD0F158295F6
		499AF8C8C33A9BFE	C138358DCECC8FC7
108		02C3F061A237BBEB	AC28B0307127EA7C
		42CBF061A637BBEB	3FF1DAED9E0FCBC5
109	**	80E529E69EDE6827	1DF1DB7B66BA1AF1
		C0ED29E69ADE6827	15700151A5804549
110		B55E84630067B8D5	88321611FF9DA421
		F55684630467B8D5	90649D7EACF91F9A
111		2749C2EBC603BFF2	A62B23A7348E2C3A
		6741C2EBC203BFF2	EB760A09C7FF5153
112	**	C4C5E14D4C5D9FF5	ABC2312FBFD94DF5
		84CDE14D485D9FF5	D2BB5954E5062D53
113	**	1566BA21F2647E18	A247ED988457CB78
		556EBA21F6647E18	5E99F231005F5249
114	**	2D093D426D922F92	5DF62030B9F23AE9
		6D013D4269922F92	5D92DA1FA3D07BA1
115		004518468E0C96C3	F28D85FF7E84F38F
		404D18468A0C96C3	52541B0443053C57
116	**	437B70A98AE03344	04B3FBF9823B4CF7
		037370A98EE03344	14EBEC79DAD3093E
117		2D01F1073D3E375B	F10B3E1EE356226C
		6D09F107393E375B	6FF26DA5E3525B62
118	*	66573DD7E0D7F110	F2F26204C29FE51E
		265F3DD7E4D7F110	083A4ECE57E429AC
119		0846DB9538155201	F120D0D2AE788057
		484EDB953C155201	00CC914A33034782
120		ABB34FC195C820D1	5F17AE066B50FC81
		EBBB4FC191C820D1	2858DD63A2FA4B53

Annexe C : <u>liste des questions posées lors de l'Interview avec Mr Vincent Rijmen</u>

1. Welke zijn de criteria voor de keuze van de S-Box (inverse + affine transformation)
2. Welke zijn deze voor de Mixing column
3. Waarom hebt U Key Schedule zo gemaakt (met S-Box een keer op elk rond) ?
4. Square Attack en differentieel aanval ?
5. Hoe hebt U de waarden van de verschillende aanvallen gemeten ?
6. Hoe bent U zeker dat er geen zwakke sleutels en geen 'trapdoor' in bestaan
7. Wat denkt-U van de Power attack ?
8. In welke maat denkt U dat de Rijndael in de toekomst gebruikt zal worden ?
9. Na de selectie wedstrijd denkt U dat andere criteria bijgevoegd moeten worden of dat gebruikt critéria geschrapt moeten worden
10. Welke zijn de grote voordelen van Uw algorithme tegenover die anderen ?
11. Na de selectie wedstrijd en nadat veel testen op Uw algorithme afgelegd geweest zijn, bent U van mening dat het algorithme nog kan verbeterd worden ?

Annexe D : Code source du rijndael en C

```
/* Rijndael
 * 8-bit assembler implementation v3.0
 */

#include <stdio.h>
#include <stdlib.h>


#define ROUNDS 14          /* number of rounds       */
#define ROOT 0x11bU        /* polynomial for generating GF(2^8) */
#define KEYLENGTH 32              /* key length, in bytes (always a
multiple of 4) (16, 24 or 32) */
#define BLOCKLENGTH 16            /* block length, in bytes (always a
multiple of 4) (16, 24 or 32) */
#define KC  (KEYLENGTH/4)           /* 4, 6 or 8 */
#define BC  (BLOCKLENGTH/4)         /* 4, 6 or 8 */

typedef unsigned short         word16;      /* 16 bit */

word16 S[256] = {

 99, 124, 119, 123, 242, 107, 111, 197,  48,   1, 103,  43, 254, 215, 171, 118,
202, 130, 201, 125, 250,  89,  71, 240, 173, 212, 162, 175, 156, 164, 114, 192,
183, 253, 147,  38,  54,  63, 247, 204,  52, 165, 229, 241, 113, 216,  49,  21,
  4, 199,  35, 195,  24, 150,   5, 154,   7,  18, 128, 226, 235,  39, 178, 117,
  9, 131,  44,  26,  27, 110,  90, 160,  82,  59, 214, 179,  41, 227,  47, 132,
 83, 209,   0, 237,  32, 252, 177,  91, 106, 203, 190,  57,  74,  76,  88, 207,
208, 239, 170, 251,  67,  77,  51, 133,  69, 249,   2, 127,  80,  60, 159, 168,
 81, 163,  64, 143, 146, 157,  56, 245, 188, 182, 218,  33,  16, 255, 243, 210,
205,  12,  19, 236,  95, 151,  68,  23, 196, 167, 126,  61, 100,  93,  25, 115,
 96, 129,  79, 220,  34,  42, 144, 136,  70, 238, 184,  20, 222,  94,  11, 219,
224,  50,  58,  10,  73,   6,  36,  92, 194, 211, 172,  98, 145, 149, 228, 121,
231, 200,  55, 109, 141, 213,  78, 169, 108,  86, 244, 234, 101, 122, 174,   8,
186, 120,  37,  46,  28, 166, 180, 198, 232, 221, 116,  31,  75, 189, 139, 138,
112,  62, 181, 102,  72,   3, 246,  14,  97,  53,  87, 185, 134, 193,  29, 158,
225, 248, 152,  17, 105, 217, 142, 148, 155,  30, 135, 233, 206,  85,  40, 223,
140, 161, 137,  13, 191, 230,  66, 104,  65, 153,  45,  15, 176,  84, 187,  22,
};

word16 rcon[30] = {
  1,2, 4, 8, 16, 32, 64, 128, 27, 54, 108, 216, 171, 77, 154, 47, 94,
188, 99, 198, 151, 53, 106, 212, 179, 125, 250, 239, 197, 145, };

word16 shifts[4][2] = {
   0, 0,
   1, 3,
   2, 2,
   3, 1};

word16 xtime(word16 a) {
   if (a & 0x80) return (a << 1) ^ ROOT;
   else return (a << 1);
   }

void MixColumn(word16 a[4][BC]) {
   word16 i, j, b[4][BC];
   printf("voor theta: \n");
```

```
   for(i = 0; i < 4; i++) {
      printf("       ");
      for(j = 0; j < BC; j++) printf("%02x ",a[i][j]);
      printf("\n");
      }
   for(j = 0; j < BC; j++)
      for(i = 0; i < 4; i++)
         b[i][j] = xtime(a[i][j])
               ^ xtime(a[(i + 1) % 4][j]) ^ a[(i + 1) % 4][j]
            ^ a[(i + 2) % 4][j]
            ^ a[(i + 3) % 4][j];
   for(i = 0; i < 4; i++)
      for(j = 0; j < BC; j++) a[i][j] = b[i][j];
   printf("na theta: \n");
   for(i = 0; i < 4; i++) {
      printf("       ");
      for(j = 0; j < BC; j++) printf("%02x ",a[i][j]);
      printf("\n");
      }
   }


void Substitution(word16 a[4][BC], word16 box[256]) {
   int i, j;
   for(i = 0; i < 4; i++)
      for(j = 0; j < BC; j++) a[i][j] = box[a[i][j]] ;
   }

void ShiftRow(word16 a[4][BC], word16 d) {
   word16 tmp[BC], i, j;
   printf("voor pi: \n");
   for(i = 0; i < 4; i++) {
      printf("       ");
      for(j = 0; j < BC; j++) printf("%02x ",a[i][j]);
      printf("\n");
      }
   /* row  0 remains unchanged */
   /* rows 1 -- 3 are shifted */
   for(i = 1; i < 4; i++) {
      for(j = 0; j < BC; j++) tmp[j] = a[i][(j + shifts[i][d]) % BC];
      for(j = 0; j < BC; j++) a[i][j] = tmp[j];
      }
   }

void KeyAddition(word16 a[4][BC], word16 rk[4][BC]) {
   int i, j;
   for(i = 0; i < 4; i++)
      for(j = 0; j < BC; j++) a[i][j] ^= rk[i][j];
   }

void keysched(word16 k[KEYLENGTH], word16 rk[ROUNDS+1][4][BC]) {
   int i, j, t, rconpointer = 0, r;
   word16 tk[4][KC];
   for(j = 0; j < KC; j++)
      for(i = 0; i < 4; i++)
         tk[i][j] = k[i+j*4];
   t = 0;
   /* copy values into round key array */
   for(j = 0; (j < KC) && (t < (ROUNDS+1)*BC); j++, t++)
      for(i = 0; i < 4; i++) rk[t / BC][i][t % BC] = tk[i][j];
   while (t < (ROUNDS+1)*BC) {
```

```
      /* calculate new values (using the round key evolution (phi))
*/
      for(i = 0; i < 4; i++)
         tk[i][0] ^= S[tk[(i+1)%4][KC-1]];
      tk[0][0] ^= rcon[rconpointer++];
if (KC != 8)
      for(j = 1; j < KC; j++)
         for(i = 0; i < 4; i++)
          tk[i][j] ^= tk[i][j-1];
else {
      for(j = 1; j < KC/2; j++)
         for(i = 0; i < 4; i++) tk[i][j] ^= tk[i][j-1];
      for(i = 0; i < 4; i++) tk[i][KC/2] ^= S[tk[i][KC/2 - 1]];
      for(j = KC/2 + 1; j < KC; j++)
         for(i = 0; i < 4; i++) tk[i][j] ^= tk[i][j-1];
     }
      /* copy values into round key array */
      for(j = 0; (j < KC) && (t < (ROUNDS+1)*BC); j++, t++)
         for(i = 0; i < 4; i++) rk[t / BC][i][t % BC] = tk[i][j];
      }
#if 1
   printf("key:\n");
   for(i = 0; i < 4; i++) {
      for(j = 0; j < KC; j++) printf("%02x ",k[i+4*j]);
      printf("\n");
      }
   printf("\n");
   printf("roundkeys:\n");
   for(r = 0; r <= ROUNDS; r++) {
      for(i = 0; i < 4; i++) {
       for(j = 0; j < BC; j++) printf("%02x ",rk[r][i][j]);
       printf("\n");
       }
      printf("\n");
      }
   printf("\tdone\n");
#endif
   }


void encrypt(word16 a[4][BC], word16 rk[ROUNDS+1][4][BC]) {
   int r, i, j;

   for(r = 0; r < ROUNDS-1; r++) {
      KeyAddition(a,rk[r]);
      Substitution(a,S);
      ShiftRow(a,0);
      MixColumn(a);
      }
   KeyAddition(a,rk[ROUNDS-1]);
   Substitution(a,S);
   ShiftRow(a,0);
   KeyAddition(a,rk[ROUNDS]);
   }


void main() {
   word16 a[4][BC], k[KEYLENGTH], i, j, rk[ROUNDS+1][4][BC];


   for(i = 0; i < 4; i++) k[i] = 0;
```

```
for(i = 4; i < 8; i++) k[i] = 0;
for(i = 8; i < 12; i++) k[i] = 0;
for(i = 12; i < 32; i++) k[i] = 0;
a[0][0] = 0;
a[1][0] = 0;
a[2][0] = 0;
a[3][0] = 0;
a[0][1] = 0;
a[1][1] = 0;
a[2][1] = 0;
a[3][1] = 0;
a[0][2] = 0;
a[1][2] = 0;
a[2][2] = 0;
a[3][2] = 0;
a[0][3] = 0;
a[1][3] = 0;
a[2][3] = 0;
a[3][3] = 0;
printf("plaintext:  \n");
for(i = 0; i < 4; i++) {
   printf("       ");
   for(j = 0; j < BC; j++) printf("%02x ",a[i][j]);
   printf("\n");
   }

keysched(k,rk);
encrypt(a,rk);

printf("ciphertext: \n");
for(i = 0; i < 4; i++) {
   printf("       ");
   for(j = 0; j < BC; j++) printf("%02x ",a[i][j]);
   printf("\n");
   }
}
```